

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Kécia Gomes de Moura

GERARD – UMA INTERFACE EDUCATIVA PARA O ENSINO DE ESTRUTURAS ADITIVAS E MULTIPLICATIVAS

Juazeiro - BA

2010

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Kécia Gomes de Moura

GERARD – UMA INTERFACE EDUCATIVA PARA O ENSINO DE ESTRUTURAS ADITIVAS E MULTIPLICATIVAS

Trabalho de Conclusão de Curso apresentado a Universidade Federal do Vale do São Francisco – UNIVASF, campus de Juazeiro, como requisito parcial para obtenção do título de Engenheiro da Computação.

Orientadora: Profª. Ana Emilia de Melo Queiroz.

Co-orientador: Prof. Ricardo Argenton Ramos

Juazeiro - BA

2010

À minha família...

AGRADECIMENTOS

Ao bom Deus que sempre guiou e iluminou meus passos nessa caminhada.

À minha mãe, Maria Lindinauva, pelo carinho e sabedoria, pela simplicidade e doçura, pelo exemplo de mãe, mulher, irmã e amiga que guardo e levo sempre comigo.

Ao meu pai, Antonio José, pelo apoio, luta e coragem que me trouxeram até aqui.

Aos meus irmãos, Bruno e Raoni, por todos os momentos de cumplicidade, por todos os sonhos e planos que sempre me impulsionaram.

À Bibia, minha segunda mãe, minha irmã, minha amiga; e aos seus filhos Heittor e Jullia.

Ao Alê...

A todos os colegas e amigos da primeira turma de engenharia da computação, e em especial à: Pat, Milla, Bacurau, Jessyka, Ivan, Felipe, Tiago Fernandes, Thiago Gerson, Deleisson e Neto.

A todos os colegas e amigos da Univasf.

A todos os professores que fazem e fizeram parte do colegiado de Engenharia da Computação pelos ensinamentos e estímulos durante todo este período de formação.

À minha orientadora, professora Ana Emilia de Melo Queiroz, pela dedicação e colaboração bem como pela qualidade profissional e prontidão no desenvolvimento deste trabalho.

Ao meio co-orientador, professor Ricardo Ramos, um exemplo de pessoa e de profissional.

E a todos aqueles que contribuíram direta ou indiretamente para minha formação profissional e pessoal.

RESUMO

Neste trabalho, é descrito o projeto e desenvolvimento de uma interface educativa para o ensino de estruturas aditivas e multiplicativas. Este ensino é realizado através da manipulação (mediada pela interface) dos diagramas de Gerard Vergnaud, propositor da Teoria dos Campos Conceituais (MARGINA et al., 2000). A utilização da notação de diagramas propostos por Vergnaud permite a diferenciação entre os diversos tipos de problemas de adição e multiplicação que existem. O aplicativo foi desenvolvido em linguagem JAVA atendendo os requisitos levantados através da prototipação em papel e modelados através de diagramas da UML (*Unified Modeling Language*).

Palavras-chave: Engenharia de Software, Software Educativo, Teoria dos Campos Conceituais, Raciocínio Matemático.

ABSTRACT

This paper describes the project and development of an educational interface in order to teaching additive and multiplicative structures. This teaching is accomplished through the manipulation (mediated by software) of Gerard Vergnaud diagrams, developer of The Theory of Conceptual Fields (MARGINA et al., 2000). Using the notation of diagrams proposed by Vergnaud allows to differentiate the various types of additive and multiplicative problems. This interface was developed in JAVA language, satisfying the requirements captured by paper prototyping and modeled using diagrams of UML (Unified Modeling Language).

Keywords: Software Engineering, Educational Software, Conceptual Fields Theory, Mathematical Reasoning.

LISTA DE FIGURAS

Figura 1. Legendas de Gerard Vergnaud. *8*
Figura 2. Diagramas das estruturas aditivas *8*
Figura 3. Diagramas das estruturas multiplicativas *9*
Figura 4. Diagrama de Composição na resolução da situação-problema a. *10*
Figura 5. Diagrama de Transformação na resolução da situação-problema b. *11*
Figura 6. Diagrama de Comparação na resolução da situação-problema c. *12*
Figura 7. Diagrama de Multiplicação *13*
Figura 8. Diagrama de Divisão por Partes. *13*
Figura 9. Diagrama de Divisão por Cotas *14*
Figura 10. Divisão da arquitetura de um sistema. *16*
Figura 11. Exemplo de uma interface com componentes GUI. *20*
Figura 12. Versões e aplicações do software Astah (ASTAH, 2011). *21*
Figura 13. Interface do ambiente Astah Community 6.3 ..... *22*
Figura 14. Interface do ambiente Netbeans 6.9.1 ..... *23*
Figura 15. Componentes do diagrama entidade-relacionamento. *25*
Figura 16. DER para uma seguradora de automóveis. *25*
Figura 17. Interface do software ERwin. *26*
Figura 18. Logomarca do SQLite *27*
Figura 19. Storyboard da utilização da interface Gerard em prototipação. *30*
Figura 20. Interface do GERARD *35*
Figura 21. Tarefas realizadas durante a modelagem do problema. *37*
Figura 22. Modelo do Diagrama de Caso de Uso *38*
Figura 23. Diagrama de Casos de Uso do Gerard. *39*
Figura 24. Modelo do Diagrama de Classe. *41*
Figura 25. Ontotologida da teoria instrumental (QUEIROZ, 2006). *42*
Figura 26. Modelo do banco de dados do software GERARD. *43*
Figura 27. Versão teste do GERARD *45*
Figura 28. Interface do GERARD desenvolvido – estruturas aditivas *47*
Figura 29. Interface GERARD desenvolvido - estruturas multiplicativas *48*
Figura 30. Itens de menu do GERARD *50*
Figura 31. Tela login do GERARD. *50*
Figura 32. Tela cadastro do GERARD. *51*
Figura 33. Tela de exibição de histórico do GERARD *52*
Figura 34. Janelas de diálogo do GERARD - (a) janela de questionamento. (b) Janela de instrução. (c) Janela de solicitação. *53*
Figura 35. MVC1 – GERARD *54*
Figura 36. MVC2 - GERARD *55*
Figura 37. Padrão de projeto Decorator - GERARD *56*
Figura 38. Padrão State - GERARD. *58*
Figura 39. Diagrama de casos de uso da interface educativa GERARD *73*
Figura 40. Organização do software *87*
Figura 41. Organização do pacote View. *88*
Figura 42. Visão expandida do pacote View. *89*
Figura 43. Organização do pacote Model *90*
Figura 44. Visão expandida do pacote Model. *91*
Figura 45. Organização do pacote Controller. *92*

Figura 46. Visão expandida do pacote Controller. *93*
Figura 47. Organização do pacote Banco. *94*
Figura 48. Visão expandida do pacote Banco. *95*
Figura 49. Organização do pacote Util. *96*
Figura 50. Visão expandida do pacote Util (1). *97*
Figura 51. Visão expandida do pacote Util (2). *98*
Figura 52. Organização do sub-pacote Util. *99*
Figura 53. Visão expandida do sub-pacote Desenhos. *100*
Figura 54. MVC 1. ..... *101*
Figura 55. MVC 2. ..... *102*
Figura 56. MVC 3. ..... *102*

LSTA DE TABELAS

Tabela 1. Botões da barra de tarefas 36
Tabela 2. Tipos de erros com respectivas mensagens exibidas para o usuário. 44
Tabela 3. Resultados encontrados a partir dos testes realizados com os usuários. 46
Tabela 4. Descrição das funções do painel de botões. 48

SUMÁRIO

1 INTRODUÇÃO 1

1.1 MOTIVAÇÃO 2
1.2 OBJETIVO GERAL 3
1.3 METODOLOGIA 3
1.4 CONTRIBUIÇÕES 4
1.5 ORGANIZAÇÃO DO DOCUMENTO 4

2 REVISÃO DA LITERATURA 5

2.1 UMA BREVE DESCRIÇÃO DA TEORIA DOS CAMPOS CONCEITUAIS 5
2.2 CATEGORIZAÇÃO DOS TIPOS DE SITUAÇÕES PROBLEMAS 6
2.3 LEGENDAS E DIAGRAMAS DE VERGNAUD 7
2.4 ESTRUTURAS ADITIVAS E SITUAÇÕES-PROBLEMAS 9
2.4.1 Composição de Medidas *10*
2.4.2 Transformação de Medidas *11*
2.4.3 Comparação de Medidas *11*
2.5 ESTRUTURAS MULTIPLICATIVAS E SITUAÇÕES-PROBLEMAS 12
2.5.1 Multiplicação *12*
2.5.2 Divisão por Partes *13*
2.5.3 Divisão por Cotas *14*

3 MATERIAIS 15

3.1 LINGUAGENS 15
3.1.1 UML - Unified Modeling Language *15*
3.1.2 Linguagem JAVA *17*
3.1.3 Java Applets *18*
3.1.4 JAVA 2D *19*
3.1.5 Java Swing *19*
3.2 AMBIENTES 21
3.2.1 Astah Community – modelagem UML *21*
3.2.2 Netbeans – codificação em Java *23*
3.2.3 Erwin – modelagem do banco de dados *24*
3.3 BANCO DE DADOS – SQLITE 27

4 METODOLOGIA E RESULTADOS 28

4.1 IDENTICAÇÃO DOS REQUISITOS 28
4.1.1 Sotryboard de utilização do GERARD *28*
4.1.2 Requisitos Verificados *33*
4.2 PROCESSO DE MODELAGEM 37
4.2.1 Diagrama de Casos de Uso *38*
4.2.2 Diagrama de classe *40*
4.2.3 Banco de dados *41*
4.2.4 Versão teste do GERARD e experimento *44*

5 O GERARD 47

5.1 INTERFACE GRÁFICA COM USUÁRIO - GUI 47

5.2 ARQUITETURA 53
5.2.1 MVC – Model View Controller *53*
5.2.2 Padrão de projeto Decorator *56*
5.2.3 Padrão de projeto State *57*
6 CONCLUSÃO 60
6.1 CONTRIBUÍÇÕES 60
6.2 TRABALHOS FUTUROS 61
7 REFERÊNCIAS 62
ANEXO A 66
ANEXO B 72
ANEXO C 86

1 Introdução

A utilização de softwares educativos como colaboradores do processo de ensino e aprendizagem, nas instituições educacionais, tem se tornado cada vez maior (BRAGA, 2006). Esta adoção está diretamente vinculada à importância de se utilizar TIC (Tecnologias da Informação e Comunicação) no contexto educacional não só como recurso didático-pedagógico, mas como base essencial para que o aluno venha a ingressar no mercado de trabalho. Segundo Miskulin et al. (2008 *apud* D'AMBROSIO, 1990, p.17), "*ignorar a presença de computadores e calculadoras é condenar os estudantes a uma subordinação total a subempregos*".

Atrelada à importância de TIC dentro das escolas, existe a necessidade de capacitação dos professores para uma visão educacional modelada em um novo processo de aprendizagem, em que o professor deixa de ser transmissor para ser mediador entre o estudante e as situações que podem favorecer aprendizagem, fazendo com que os alunos se tornem agentes na construção dos seus próprios conhecimentos (NASCIMENTO e NASCIMENTO, 2009). Dessa forma, cabe ao profissional de educação verificar a qualidade do *software* que se pretende utilizar, exigindo que o mesmo ofereça recursos concordantes com a concepção de aprendizagem construtivista, isto é, que permite a construção do conhecimento a partir de percepções e ações do próprio usuário (GRAVINA e SANTAROSA,1998).

No âmbito do ensino e aprendizagem da matemática, existem muitos aplicativos e jogos interativos que primam por uma aprendizagem construtivista. Porém, muitos destes apenas conseguem mostrar para os alunos as fórmulas e teorias já encontradas nos livros, instruindo-os apenas à prática de memorização e do pensamento mecânico para resolução de problemas. (GRAVINA e SANTAROSA,1998). Segundo Gravina e Santarosa (1998), essa forma de apresentação de conceitos não desafia a capacidade cognitiva dos alunos, as autoras enfatizam que:

> No contexto da Matemática, a aprendizagem nesta perspectiva depende de ações que caracterizam o 'fazer matemática': experimentar, interpretar, visualizar, induzir, conjeturar, abstrair,

> generalizar e enfim demonstrar. É o aluno agindo, diferentemente de seu papel passivo frente a uma apresentação formal do conhecimento, baseada essencialmente na transmissão ordenada de 'fatos', geralmente na forma de definições e propriedades.

Corroborando a assertiva de Gravina e Santarosa (1998), Miskulin et al. (2008) alude que a Matemática deve ser mediada, não por modelos obsoletos, que não contribuem de modo significativo para o desenvolvimento e transformação do indivíduo, mas por metodologias que façam sentido e tenham relação com a sua integração na sociedade. Ainda segundo o autor, sem uma educação matemática, com qualidade, o indivíduo talvez seja privado de crescer no saber matemático, sendo este (o saber matemático) importante para a qualificação profissional em qualquer área.

1.1 Motivação

De acordo com os indicadores nacionais (INEP, 2007) há uma grande deficiência dos alunos na aprendizagem de matemática, sobretudo no Ensino Fundamental (momento em que aluno adquire conhecimentos básicos e importantes para sua vida futura). De acordo com Gomes et al. (2004) alunos da 4ª série demonstram apenas capacidade de resolver problemas de subtração e adição, situações cotidianas e reconhecimento de figuras geométricas simples.

Segundo MARGINA et al. (2000), o ensino da matemática depende de concepções sobre a natureza do conhecimento matemático e de como acontece o desenvolvimento cognitivo do ser humano. Daí a importância do professor como mediador do processo de aquisição do conhecimento matemático. De acordo com Gomes et al. (2008 *apud* Vergnaud, 1991), "*existem vários tipos de relações aditivas, e em conseqüência, vários tipos de (problemas de) adição e subtração*". Portanto, durante o ensino da matemática é preciso que o professor atente para os diversos tipos de situações problemas, de forma a não ficar repetindo problemas que exigem um único raciocínio lógico do aluno. Para resolução das diversas situações-problemas

existentes, Vergnaud propôs a utilização de diagramas, pois, segundo ele, a utilização de diagramas auxilia na sistematização do problema durante o cálculo relacional (MARGINA et al., 2000). Margina et al. (2000), ainda enfatiza a importância de conhecer os conceitos descritos por Vergnaud:

> Temos a convicção que o professor ao entender melhor a aquisição desses conceitos estará melhor instrumentalizado para introduzi-los e desenvolvê-los. Como conseqüência direta haverá, por parte do professor, uma melhoria de sua prática docente e, por parte do aluno, um melhor desempenho na realização de suas tarefas.

Sendo assim, a importância de *softwares* educativos, a relevância do papel do professor enquanto mediador do conhecimento para o aluno e a deficiência verificada na aprendizagem da matemática motivam o desenvolvimento do presente trabalho.

1.2 Objetivo Geral

O principal objetivo deste presente trabalho é desenvolver uma interface educativa para ensino de estruturas aditivas e multiplicativas.

Para atingir este objetivo, foi utilizada a seguinte metodologia:

1.3 Metodologia

- Revisão bibliográfica sobre Linguagem Java, Applets Java, Java 2D, métodos ágeis de desenvolvimento, padrões de projeto, banco de dados, prototipação e leitura dos referenciais teóricos na área de padrões de projeto para desenvolvimento de interfaces educativas;
- Geração de um documento de requisitos;

- Implementação em linguagem JAVA (ORACLE, 2010) considerando os casos de uso e os padrões de projeto adotados na literatura sobre interfaces educativas;
- Teste do aplicativo desenvolvido com os professores;
- Disponibilização do aplicativo para o uso nas escolas de Juazeiro-BA.

1.4 Contribuições

O estado da Bahia é um dos maiores do Brasil e apresenta longas distâncias entre os municípios do interior e os grandes centros. Nesse sentido, a interface aqui refinada será utilizada em programas de educação à distância, favorecendo que professores que estejam afastados dos grandes centros possam ter acesso a interfaces computacionais voltadas para aprendizagem de conceitos específicos.

1.5 Organização do Documento

Este documento está organizado da seguinte forma: no Capítulo 2 é descrita de forma sucinta e objetiva a Teoria dos Campos Conceituais de Verganud, bem como a categorização dos tipos de situações-problemas e a utilização dos diagramas para resolução dos mesmos. No Capítulo 3 são descritas as linguagens e os ambientes computacionais que foram utilizados para modelar e implementar a interface Gerard. No Capítulo 4, é apresentada a metodologia e os resultados obtidos durante o desenvolvimento. No capítulo 5, é apresentado o GERARD desenvolvido descrevendo a interface e a arquitetura do mesmo. No Capítulo 6, são feitas as considerações finais evidenciando a contribuição do trabalho desenvolvido.

2 Revisão da Literatura

2.1 Uma breve descrição da Teoria dos Campos Conceituais

Nesta seção, será realizada uma descrição dos principais aspectos da Teoria dos Campos Conceituais. Obviamente não se pretende aqui destrinchar e discutir detalhadamente o que Gérard Vergnaud propôs depois de longos anos de estudos, mas evidenciar a importância da teoria para o ensino e aprendizagem da matemática uma vez que a mesma nos leva à percepção de como os alunos constroem o conhecimento matemático.

> De fato esta teoria, ao congregar com êxito a Psicologia Cognitiva e a Matemática, vem se tornando uma das mais expressíveis no campo da Educação Matemática. Suas idéias a muito tem ajudado os pesquisadores a entender a formação e desenvolvimento de conceitos matemáticos dos alunos a partir da observação de suas estratégias de ação. E, ainda, não foi por outro motivo que ela ganhou o status de ser uma das principais teorias sobre a qual se apóia os Parâmetros Curriculares Nacionais – PCNs – de Matemática. O que vem mais uma vez reforçar a necessidade de um produto como este. (MARGINA et al., 2000).

Segundo Margina et al. (2000 *apud* Vergnaud, 1986) a aprendizagem acontece por meio das experiências com um grande número de situações, tanto dentro como fora da escola. Os conceitos adquiridos possuem um domínio de validade restrito que varia de acordo com a experiência e com o desenvolvimento cognitivo do aluno. Por exemplo, ao se multiplicar 8 por 2 será obtido 16, e se agora multiplicarmos 8 por um valor maior que 2 o produto será ainda maior que 16, formando-se assim a concepção de que “multiplicação sempre aumenta”. Porém, esta idéia se restringe apenas ao domínio dos números naturais. (MARGINA et al., 2000).

> Aprender um conceito matemático, portanto, implica dominar um conjunto de propriedades que emergem diferentes situações e que

> são mediadas por diferentes sistemas de representações. Dominar um campo conceitual significa saber resolver problemas em situações diversas nas quais determinado conceito está inserido. (GRAVINA e SANTAROSA,1998).

Um exemplo claro é o de uma criança que para contar bolinhas de gude precisa dispô-las em uma fila. Porém com apenas este conhecimento a criança terá dificuldades de contar quando a soma dos objetos possuir um valor demasiadamente grande, tornando impossível a disposição destes em uma fila. Este exemplo mostra que o conceito de contagem precisa de diferentes situações para emergir. (MARGINA et al., 2000).

> Esta é justamente uma das razões para estudarmos a Teoria dos Campos Conceituais, uma vez que os conceitos matemáticos traçam seus sentidos a partir de uma variedade de situações e que cada situação normalmente não pode ser analisada com a ajuda de um único conceito. Em outras palavras, nem um conceito nem uma situação isoladamente dá conta do processo de aquisição de um conhecimento. É por este motivo que nós, em sintonia com Vergnaud, propomos estudar os conceitos matemáticos não como conceitos isolados, mas como conjuntos de conceitos inter-relacionados com conjuntos de situações. (MARGINA et al., 2000).

2.2 Categorização dos tipos de situações problemas

Apesar de intuitivamente inferirmos a existência de uma vasta quantidade de problemas ligados à adição, subtração, multiplicação e divisão, Vergnaud propôs a existência de ‘apenas’ 6 (seis) classes fundamentais de problemas ligados adição. Essas situações são: composição de medidas, transformação de medidas, relação de medidas, composição de transformação, composição de relação e transformação de relação (GOMES et al., 2004).

Da mesma forma, Vergnaud classificou as estruturas multiplicativas em: multiplicação, divisão por partes, divisão por cotas e quarta proporcional. Também podemos enquadrar as situações que envolvem operações com

números decimais e frações como parte das estruturas multiplicativas (GOMES et al., 2004).

Para o desenvolvimento deste trabalho, foram abordadas três classes de estruturas aditivas e três classes de estruturas multiplicativas, em outras palavras, a interface GERARD desenvolvida contempla as seguintes classes

Aditiva:

- Composição de Medidas,
- Transformação de Medidas,
- Comparação de Medidas.

Multiplicativa:

- Multiplicação,
- Divisão por Partes,
- Divisão por Cotas.

2.3 Legendas e Diagramas de Vergnaud

Para resolução de cada situação-problema referente a uma determinada classe (Seção 2.2), Gérard Vergnaud propõe a utilização de um diagrama específico para a respectiva modelagem. Os diagramas são formados um conjunto de símbolos e formas geométricas. Estes são apresentados na Figura 1.

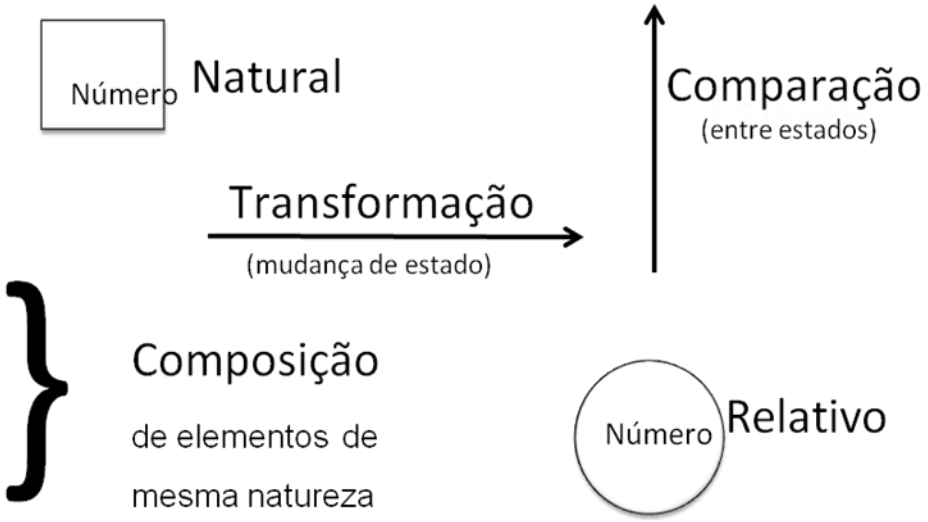


Figura 1. Legendas de Gerard Vergnaud.

De acordo com a Figura 1, o número natural é representado por um quadrado e o número relativo por um círculo. As classes de estruturas aditivas, Composição, Transformação e Comparação de medidas, são representadas, respectivamente, pelo símbolo abre chave, seta para direita e seta para cima.

A junção das legendas é formada para construir os diagramas de forma a modelar e solucionar situações-problemas relacionadas a uma determinada classe (Seção 2.2). As Figura 2 e Figura 3 apresentam os diagramas que foram contemplados na interface desenvolvida.

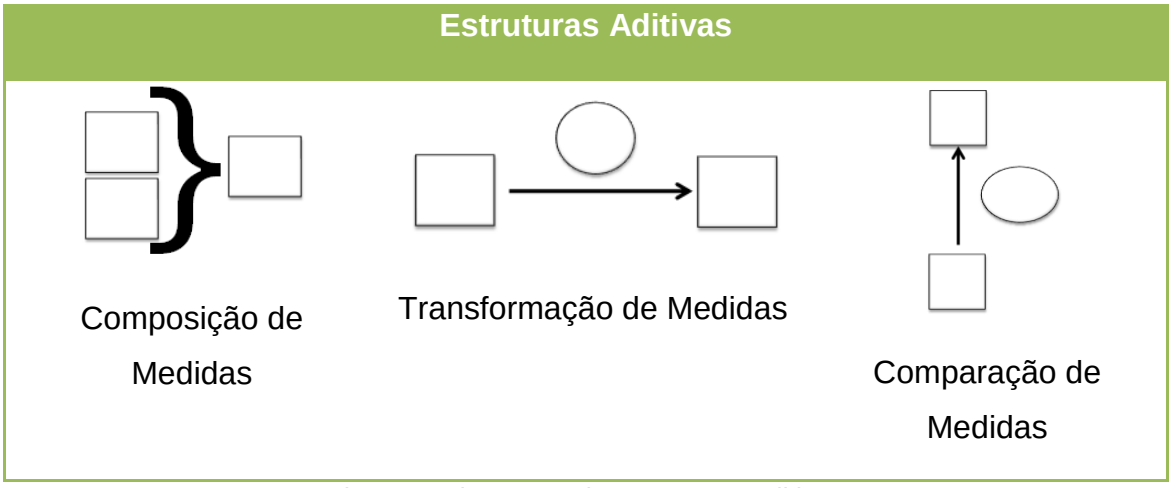


Figura 2. Diagramas das estruturas aditivas

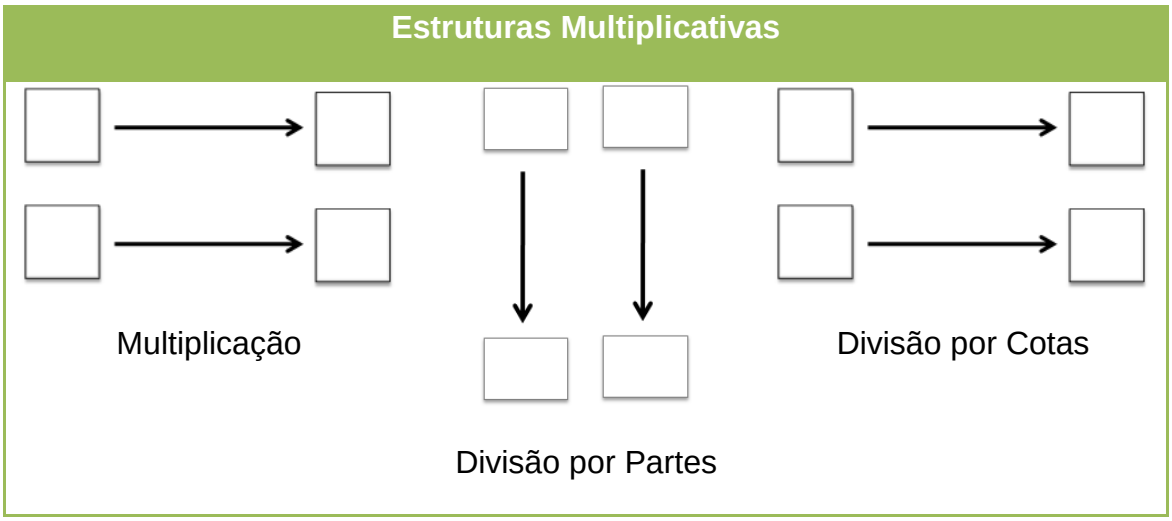


Figura 3. Diagramas das estruturas multiplicativas

2.4 Estruturas Aditivas e Situações-Problemas

De acordo com Margina et al. (2000), para dominar as estruturas aditivas, o aluno precisa ter a capacidade de resolver diversos tipos de situações-problema correlatas, não sendo suficiente apenas resolver um cálculo numérico, pois, por exemplo, por trás de um simples 4 + 7 pode-se encontrar problemas tão sofisticados que até alunos da 4ª série (aproximadamente 10-11 anos de idade) apresentam dificuldades na hora de resolver.

A variedade de situações-problemas será aqui ilustrada através de três exemplos de problemas:

Situação-problema a: Ao redor da mesa da sala de jantar de minha casa, estão sentados apenas 4 garotos e 7 garotas. Quantas pessoas estão sentadas ao redor da mesa?

Situação-problema b: Maria comprou uma caixa de bombons por R$ 4,00 e ainda ficou com R$ 7,00. Quanto ela possuía antes de fazer a compra?

Situação-problema c: Carlos tem 4 anos. Maria é 7 anos mais velha que Carlos. Quantos anos Maria tem?

Depois de analisar os problemas **a**, **b** e **c**, constata-se que para resolvê-los basta apenas fazer a adição simples "4+7". No entanto, segundo Margina et. al. (2000) a capacidade de resolver estes problemas varia de acordo com a idade e a escolaridade da criança. Isto porque, embora os três problemas se refiram à utilização de uma mesma operação aritmética, eles estão associados à idéias diferentes, o que exige raciocínios diferentes do aluno para um mesmo domínio.

Nesta seção, serão descritas as classes de estruturas aditivas (Composição, Transformação e Comparação) ilustrando a utilização do diagrama correspondente para resolução dos problemas **a**, **b** e **c**.

2.4.1 Composição de Medidas

Segundo Margina et al. (2000), a classe de problemas de Composição compreende as situações que envolvem parte e todo, isto é, juntar uma parte com outra parte para obter o todo, ou subtrair uma parte do todo para obter a outra parte. A situação-problema **a** é um caso de composição em que, por exemplo, a quantidade de garotos em uma mesa é uma parte, a quantidade de garotas é outra parte e a soma dessas duas partes (quantidade de garotos + quantidade de garotas) formam o todo das pessoas em volta da mesa. A Figura 4 ilustra a utilização do diagrama correspondente à Composição na resolução do problema **a**.

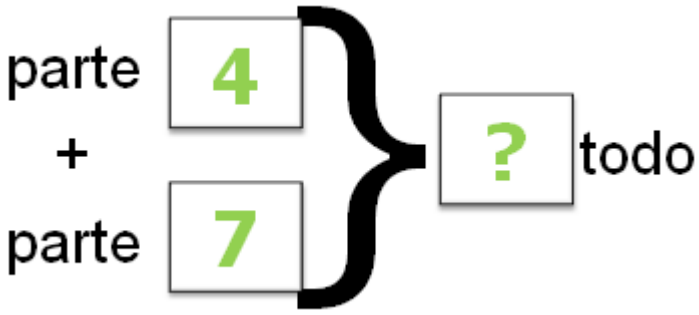


Figura 4. Diagrama de Composição na resolução da situação-problema a.

2.4.2 Transformação de Medidas

Segundo Margina et al. (2000), a classe dos problemas de Transformação é aquela que trata de situações onde a idéia temporal está sempre envolvida, isto é, no estado inicial tem-se uma quantidade que se transforma (com perda/ganho; acréscimo/decréscimo; etc.) chegando ao estado final com outra quantidade. O problema **b** é um exemplo de uma situação aditiva envolvendo transformação, em que Maria gastou R$ 4,00 (fez uma transformação da quantia que tinha), sobrou R$ 7,00 em sua carteira (o estado final do dinheiro em sua carteira) e pergunta-se pela quantia que ela tinha inicialmente (estado inicial). A Figura 5 ilustra a utilização do diagrama correspondente à Transformação na resolução do problema **b**.



Figura 5. Diagrama de Transformação na resolução da situação-problema b.

2.4.3 Comparação de Medidas

Segundo Margina et al. (2000), a classe dos problemas de comparação diz respeito aos problemas que comparam duas quantidades, sendo uma denominada referente e a outra referido. No problema **c**, é dada a idade de Carlos (4 anos) e a idade de Maria é apontada em relação à idade de Carlos (ela é 7 anos mais velha que ele). Portanto, a idade de Carlos é a referência (o referente) para se obter a idade de Maria que, neste caso, é o referido. A Figura 6 ilustra a utilização do diagrama correspondente à Comparação na resolução do problema **c**.

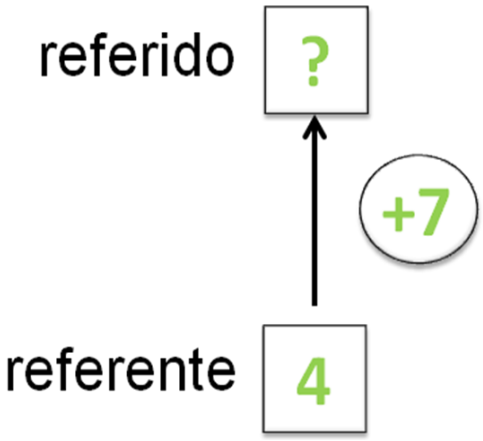


Figura 6. Diagrama de Comparação na resolução da situação-problema c.

2.5 Estruturas Multiplicativas e Situações-Problemas

Da mesma forma que para as estruturas aditivas, o ensino das estruturas multiplicativas se dá por meio da identificação e classificação de situações-problema. Após a identificação, acontece a posterior coleta de dados sobre os procedimentos e os caminhos utilizados pelos alunos para expressar seu raciocínio (VERGNAUD, 1988). Em seus estudos, Vergnaud identificou alguns tipos de situações-problema que envolvem operações de multiplicação e divisão. Nesse projeto, foram abordadas as situações multiplicativas simples: multiplicação, divisão por cotas e divisão por partes.

2.5.1 Multiplicação

Observe a situação-problema **d**:

Um bolo leva 5 ovos. Quantos ovos são necessários para fazer 7 bolos?

Neste tipo de situação, desconhece-se o valor total e conhece-se o número de partes e o valor de cada parte. Nesse caso, o valor total é encontrado multiplicando-se o número de partes pelo valor de cada parte. A Figura 7 ilustra a utilização do diagrama correspondente à Multiplicação na resolução do problema **d.**

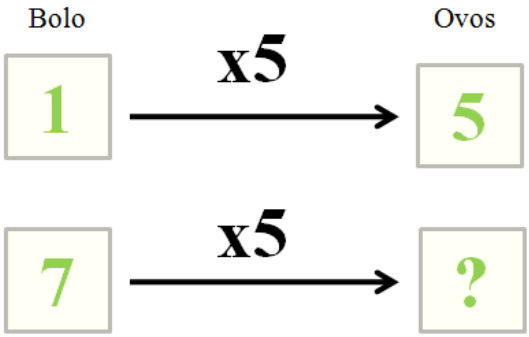


Figura 7. Diagrama de Multiplicação

2.5.2 Divisão por Partes

Observe a situação-problema **e**:

Para fazer sete bolos, são necessários trinta e cinco ovos. Se todos os bolos têm a mesma receita, quantos ovos são necessários para fazer somente um bolo?

Neste tipo de situação, desconhece-se o valor de cada parte e conhece-se o valor total e o número de partes. Nesse caso, o valor de cada parte é encontrado dividindo-se o valor total pelo número de partes. A Figura 8 ilustra a utilização do diagrama correspondente à Divisão por Partes na resolução do problema **e**.

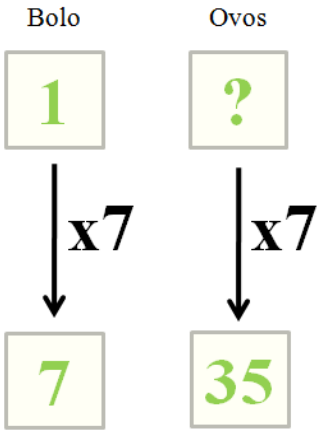


Figura 8. Diagrama de Divisão por Partes.

2.5.3 Divisão por Cotas

Observe a situação-problema **f**:

Quantos bolos podem ser feitos com trinta e cinco ovos, se cada bolo leva cinco ovos?

Nesse tipo de situação, desconhece-se o número de partes e conhece-se o valor total e o valor de cada parte. Nesse caso, o número de partes é encontroado dividindo-se o valor total pelo valor da cada parte. A Figura 9 ilustra a utilização do diagrama correspondente à Divisão por Cotas na resolução do problema **f**.

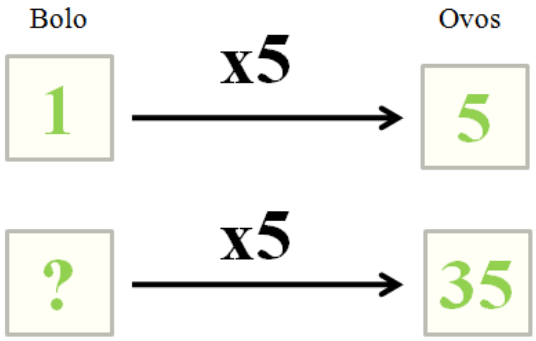


Figura 9. Diagrama de Divisão por Cotas

3 Materiais

Para desenvolvimento deste projeto foi necessária a utilização de algumas ferramentas. Estas tiveram como finalidade viabilizar o processo de construção da interface educativa desde o processo de levantamento de requisitos até a implementação.

3.1 Linguagens

Duas linguagens foram utilizadas no desenvolvimento da interface Gerard: a UML para modelagem do sistema e a linguagem JAVA para implementação. Ambas são descritas nas próximas seções.

3.1.1 UML - *Unified Modeling Language*

A UML é uma linguagem gráfica padrão para a elaboração da estrutura de projetos de software. Ela pode ser empregada para visualizar, especificar, construir e documentar os artefatos de sistemas de software. Em 1997, a UML foi adotada pela OMG (*Object Management Group*) e desde então se tornou o padrão da indústria de software para a modelagem de objetos e componentes (RUMBAUGH et al., 1999).

3.1.1.1 Visões da UML

Segundo Rumbaugh et al. (1999), a arquitetura de um sistema pode ser descrita através de *views* (visões). Cada visão constitui uma projeção na organização e estrutura do sistema, cujo foco está voltado para determinado

aspecto desse sistema. A arquitetura de um sistema pode ser descrita através de cinco visões interligadas:

- Visão de caso de uso: focaliza os comportamentos de um sistema devendo ser transparente a todos os envolvidos: gerentes, analistas, programadores e usuários finais.
- Visão de Projeto: focaliza a estrutura de um sistema através da definição de classes, colaborações e as interfaces do sistema.
- Visão de Processo: focaliza as questões de desempenho e escalabilidade do sistema.
- Visão de Implementação: focaliza os artefatos físicos (programas, bibliotecas, banco de dados) para a efetiva montagem do sistema.
- Visão de Implantação: focaliza a topologia do hardware, liberação e instalação do sistema.



Figura 10. Divisão da arquitetura de um sistema.

3.1.1.2 Diagrams da UML

A UML disponibiliza diagramas específicos para a modelagem gráfica das cinco visões descritas na seção anterior:

- Diagrama de casos de uso para ilustrar as interações do usuário com o sistema
- Diagrama de classe para ilustrar a estrutura lógica
- Diagrama de objetos para ilustrar os objetos e suas interações
- Diagrama de estados para ilustrar comportamentos
- Diagrama de componentes para ilustrar a estrutura física do software
- Diagrama de Interações: composto de diagrama de seqüência e diagrama de colaboração. Utilizado para ilustrar comportamentos
- Diagrama de Atividades para ilustrar o fluxo dos eventos.

Para modelar a arquitetura do GERARD, foram utilizados os diagramas de casos de uso e de classe apresentados na Seção 4.

3.1.2 Linguagem JAVA

A diversidade de plataformas computacionais bem como de sistemas operacionais existentes, quer seja nos laboratórios das instituições de ensino ou na residência dos usuários, exige da aplicação educacional flexibilidade para se adequar aos recursos disponíveis nos ambientes onde a mesma será utilizada. Uma forma de suprir esta necessidade é a utilização, durante o desenvolvimento, de ferramentas que viabilizem a construção de programas multiplataformas, como a linguagem JAVA.

3.1.3 Java Applets

Utilizando-se a linguagem Java é possível implementar Applets. Estes são aplicativos JAVA executados na máquina do cliente em um *web browser,* por exemplo, o Internet Explorer. Applets são usados primariamente para disponibilizar a execução de um aplicativo em uma máquina diferente daquela onde está hospedado o seu código fonte. Em outras palavras, o software pode ser hospedado em um servidor nos Estados Unidos e, no entanto, ser utilizado aqui, na cidade de Juazeiro BA, por exemplo. Essa flexibilidade traz implicações tanto na disponibilidade do software quanto na sua execução. No primeiro caso, os applets favorecem que usuários afastados dos servidores possam executá-lo e utilizá-lo. Essa nova possibilidade contribui na construção de interfaces de softwares voltadas para uso a distância. No segundo caso, a execução do applet ocorre na máquina do usuário, ou seja, aquele que irá usar os recursos do software para atender suas necessidades. A execução na máquina do usuário, ou máquina cliente, contribui para evitar a sobrecarga dos servidores onde o software fica hospedado.

Um formato muito conhecido na internet é HTML (*Hyper-Text Markup Language*). Ela é uma linguagem de formatação de texto, logo apresenta algumas limitações no que diz respeito ao processo interativo entre o usuário e máquina. Os applets, por sua vez, são softwares desenvolvidos em linguagem de programação, e executados em navegadores, logo, eles fornecem recursos interativos para aplicações web que não podem ser fornecidas apenas pelo HTML (*Hyper-Text Markup Language*). Eles possibilitam a captura da entrada do mouse como, por exemplo, a rotação de objetos em 3D, e também têm controles como botões ou caixas de seleção. Em resposta à ação do usuário um applet pode alterar o conteúdo da interface. A interatividade com o usuário, a fácil programação e a sua natureza voltada para internet faz com que Java applet seja apropriado para o desenvolvimento de interfaces educativas assim como a do GERARD.

3.1.4 JAVA 2D

O Java 2D é uma API (*Application Programming Interface*) extensa e poderosa que inclui diversos tipos de funções para desenhar diretamente na tela, desenhar texto e manipular imagens. Ela permite a manipulação bem como a construção de desenhos geométricos, apresentando como principais aspectos: grande número de primitivas geométricas (retângulos, elipses, curvas), e um mecanismo para se desenhar qualquer forma geométrica; mecanismos para detectar colisão em texto e imagens; modelos de composição, que permitem escolher como combinar imagens que se interceptam; modelo de cores avançado; diretivas para controlar a qualidade do desenho; classes para manipulação uniforme dos principais formatos de imagem, etc. (DEITEL e DEITEL, 2005).

Na biblioteca Java2D, destaca-se a classe *Graphics2D* que é o principal responsável pela a impressão das imagens na tela do computador. Cada objeto gráfico tem o seu próprio método de desenho, chamado normalmente de *paintComponent*. Este método é chamado automaticamente quando o componente precisa ser desenhado ou redesenhado na tela. O procedimento padrão para desenhar gráficos com o Java2D é sobrescrever o método *paintComponent*. (DEITEL e DEITEL, 2005).

Para desenhar os diagramas, bem como produzir o efeito de *drag and drop* durante a utilização do GERARD, foi utilizado a API Java2D em conjunto com Java Swing descrito na próxima seção.

3.1.5 Java Swing

A maioria dos softwares desenvolvidos possuem uma interface similar com botões (“ok”, “cancelar”), barra de menus (“Arquivo”, “Ferramentas”), itens de menu (“Editar”, “Salvar”, “Abrir”), campos de texto, rótulos entre outros. Estes artefatos são utilizados em conjunto para produzir o que chamamos de Interface Gráfica com o Usurário – GUI (*Graphical User Interface*).

Como próprio nome sugere, a GUI é desenvolvida para permitir a interação do usuário com o software. Ela fornece a diferentes aplicativos um conjunto consistente de componentes (botões, menus...) intuitivos que entregam ao usuário um nível básico de familiaridade com um programa mesmo sem ele nunca ter utilizado-o antes (DEITEL e DEITEL, 2005). Um exemplo comum de uma GUI é apresentado na Figura 11.



Figura 11. Exemplo de uma interface com componentes GUI.

A linguagem Java possui um pacote que fornece classes para a criação de componentes GUI, chamado Swing. Este pacote se tornou padrão em Java a partir da versão 1.2 da plataforma Java 2 (ORACLE, 2010). A maior parte dos componentes Swing são escritos, manipulados e exibidos completamente em Java (conhecidos como componentes Java puros) (DEITEL e DEITEL, 2005). Em outras palavras, esses componentes não estão diretamente associados com os recursos da interface gráfica com o usuário da plataforma local, o que possibilita que os mesmos tenham a mesma aparência quando executados em diferentes plataformas Java.

Outra característica importante do Swing é o modelo de tratamento de eventos. As GUIs são baseadas em eventos, isto é, elas geram eventos quando o usuário do programa interage com a interface. Algumas interações

comuns são: mover o mouse, clicar no mouse, clicar em um botão, digitar um campo de texto, selecionar o item de um menu, fechar uma janela, etc.

Para o desenvolvimento do GERARD foram utilizados componentes do Java Swing para criação da interface gráfica com o usuário, sendo utilizados componentes como botões, rótulos e campo de texto.

3.2 Ambientes

Para modelagem da interface, foram utilizadas algumas ferramentas CASE e um ambiente de desenvolvimento para codificação do programa descritos nas próximas seções.

3.2.1 *Astah Community* – modelagem UML

Para gerar os diagramas da UML utilizados para modelar a interface GERARD, foi utilizada inicialmente a versão *trial* do *Software Enterprise Architect.* No entanto, esta versão limita o período de uso do *software* de forma gratuita. Para substituir o *Enterprise Architect,* foi adotado o *Software Astah* cuja versão *Community* é gratuita, de tempo ilimitado e abrange os diagramas da UML, como pode ser observado Figura 12.

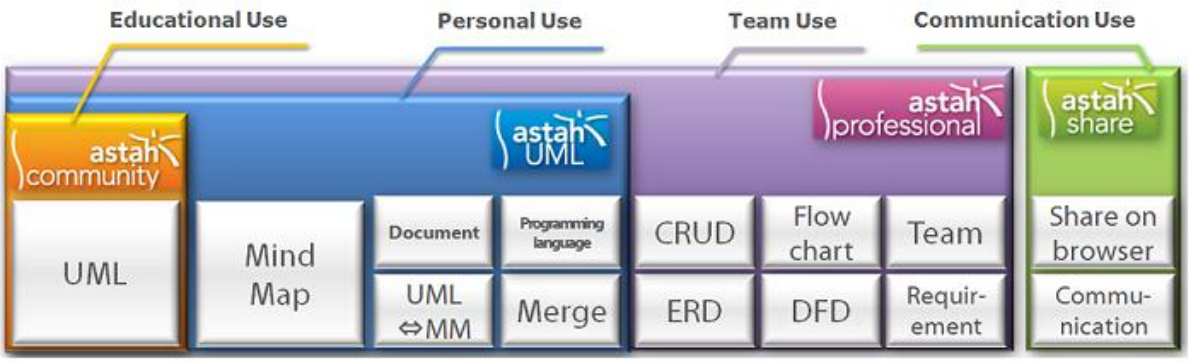


Figura 12. Versões e aplicações do software Astah (ASTAH, 2011).

Dentre os recursos da ferramenta estão:

- Suporte a UML 2.1
- Diagramas de Classe, Caso de Uso, Sequência, Atividade, Comunicação, Máquina de Estado, Componentes, Implantação, Estrutura de Composição, Objetos e Pacotes.
- Ajustes de alinhamento e tamanho dos diagramas
- Impressão dos diagramas (com a marca d'água da ferramenta)
- Exportação das imagens dos diagramas (com a marca d'água da ferramenta)

Na Figura 13, é apresentada a interface do ambiente *Astah Community* 6.3 utilizado para modelagem do GERARD.

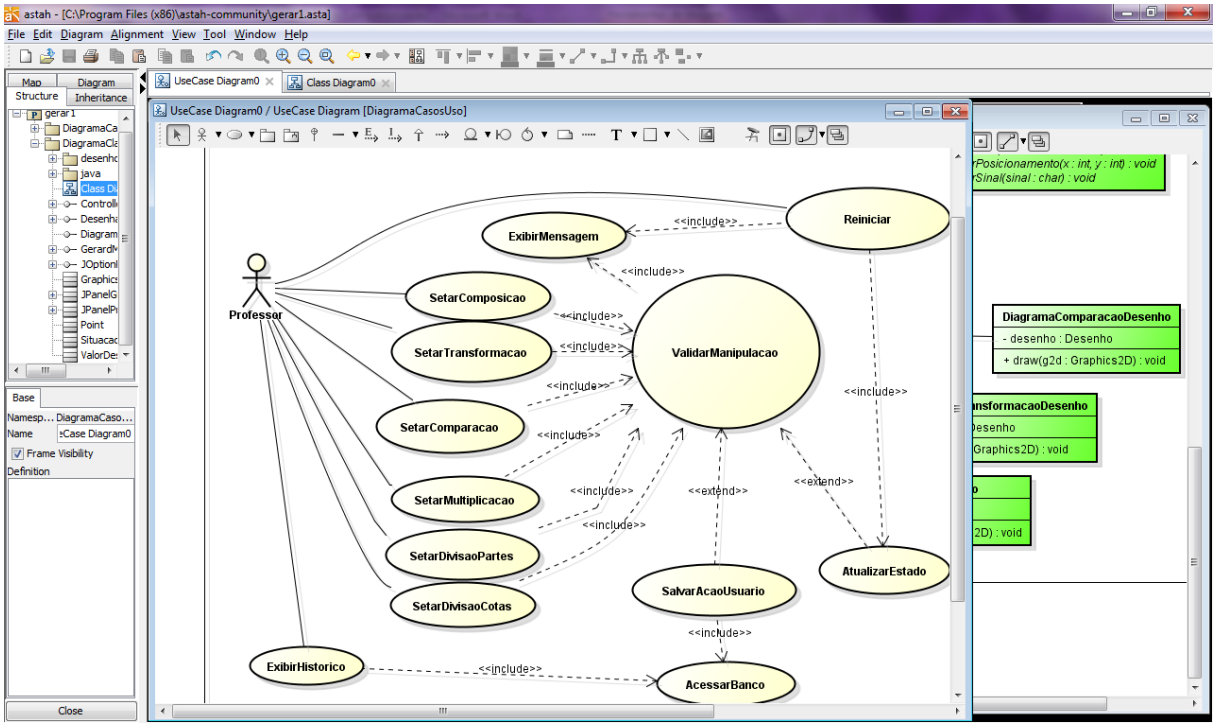


Figura 13. Interface do ambiente Astah Community 6.3

3.2.2 Netbeans – codificação em Java

Para implementação e construção da Interface Gerard, foi utilizada a IDE (*Integrated Development Environment*) Netbeans versão 6.9.1.

O NetBeans é um ambiente de desenvolvimento integrado, gratuito e de código aberto para desenvolvedores de software. Ele oferece as ferramentas necessárias para criar aplicativos profissionais de desktop, empresariais, web, móveis e multiplataformas. Este ambiente de desenvolvimento é totalmente implementado em JAVA, possuindo uma arquitetura de aplicativo confiável. Além do mais, por possuir uma arquitetura modular, o Netbeans permite o desenvolvimento de aplicações de maneira organizada e estruturada, o que torna possível o crescimento de maneira incremental do *software*. (NETBEANS, 2010). A interface de utilização do Netbeans pode ser observada na Figura 14.

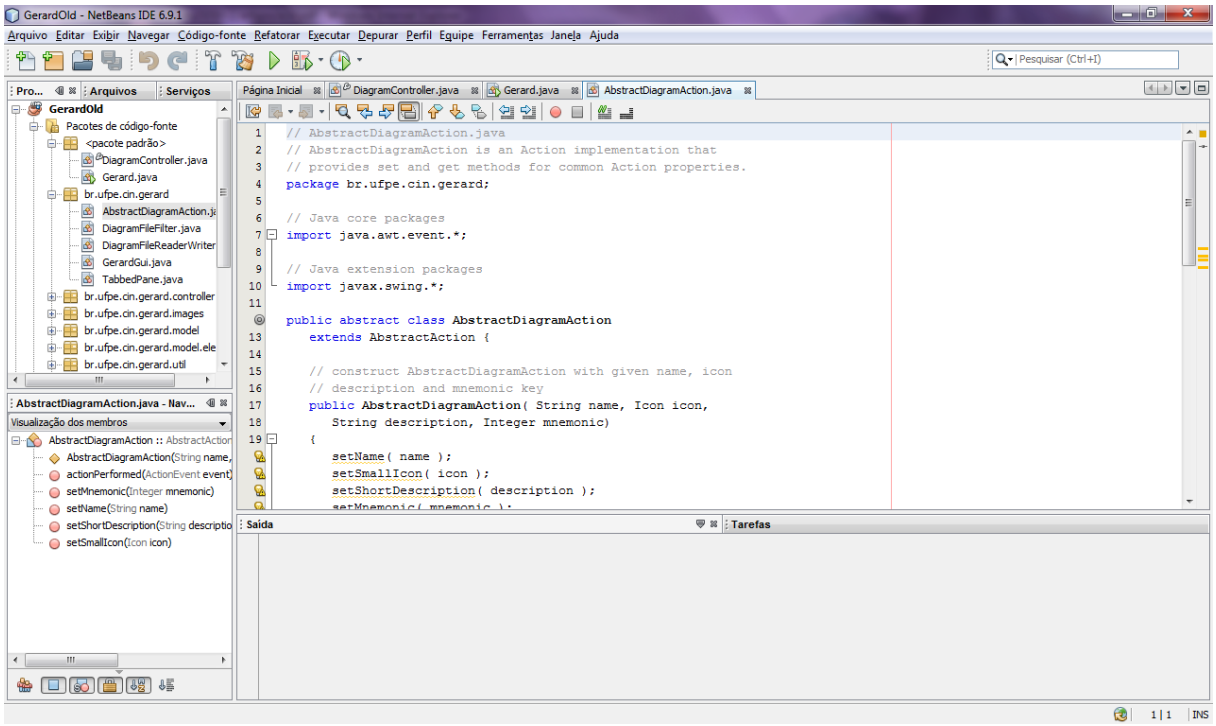


Figura 14. Interface do ambiente Netbeans 6.9.1

3.2.3 Erwin – modelagem do banco de dados

Durante o desenvolvimento do GERARD, a funcionalidade de exibição de histórico das ações realizadas por um usuário, ao utilizar a interface, foi acrescentada ao software. Para isto, foi necessária a modelagem e criação de um banco de dados.

Para projeto conceitual de aplicações de um banco de dados, normalmente é utilizado o modelo Entidade-Relacionamento (ER) que é um modelo de dados conceitual de alto nível, além de muito popular. O modelo ER é descrito através da notação diagramática associada ao modelo ER, conhecida por diagramas ER (ELMASRI e NAVATHE, 2005).

O Diagrama Entidade Relacionamento (DER) é composto por um conjunto de artefatos gráficos que visa representar todos os objetos do modelo Entidade Relacionamento tais como: atributos, atributos chaves, relacionamentos, restrições estruturais, entre outros (SANTOS, 2011).

O diagrama ER fornece uma visão lógica do banco de dados, fornecendo um conceito mais generalizado de como estão estruturados os dados de um sistema. Os objetos que compõem o diagrama ER estão listados a seguir, na Figura 15.

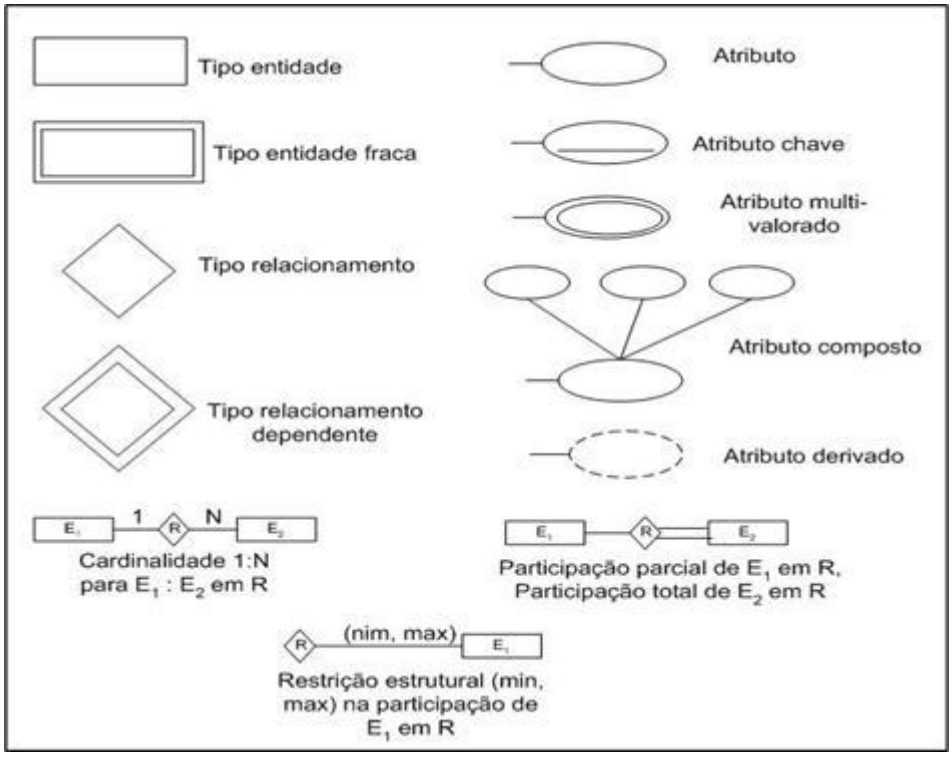


Figura 15. Componentes do diagrama entidade-relacionamento.

Um exemplo de aplicação do diagrama ER pode ser visualizado na Figura 16. Este exemplo mostra o DER para uma seguradora de automóveis, em que um cliente tem pelo menos um carro e um carro pertence a um único cliente, além disso, cada carro possui um número de acidentes associados a ele.

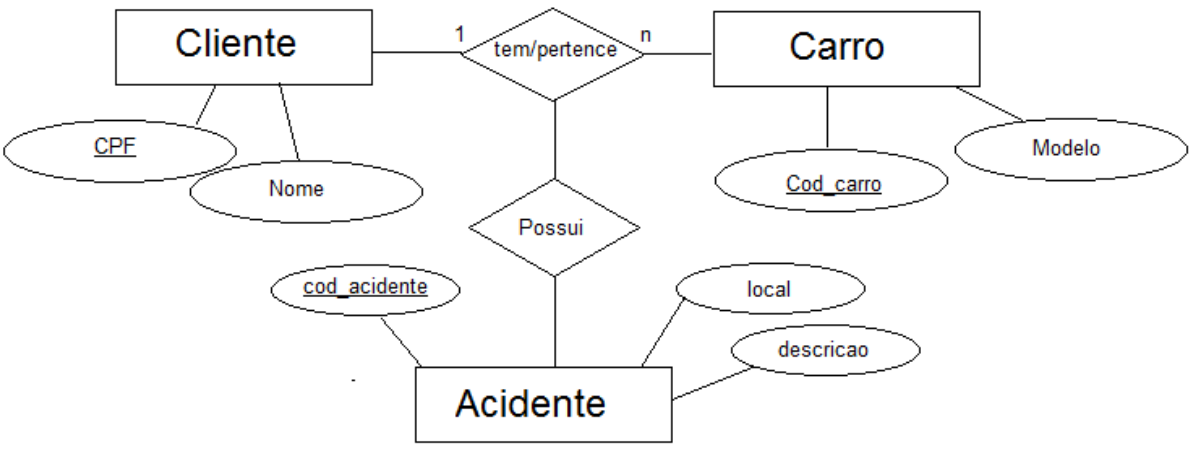


Figura 16. DER para uma seguradora de automóveis.

Para agilizar o processo de construção dos diagramas de entidade relacionamento, é utilizada uma ferramenta CASE. Neste trabalho, optou-se pelo uso do software ERwin versão 4.1. Entre outras características o ERwin:

- Possibilita uma maior concentração na organização lógica dos dados e na especificação das restrições.
- Facilita manutenção do Banco de Dados.
- Evita alguns erros de usuário na definição do esquema.

O Erwin possui duas formas de iteração durante a modelagem dos bancos de dados:

- **Nível lógico**: preocupa-se mais com os conceitos e formas de organização lógica dos dados.
- **Nível físico**: apresenta mais detalhes sobre o DER com informações mais específicas sobre o banco de dados escolhido.

A Figura 14 apresenta a interface da ferramenta ERwin.

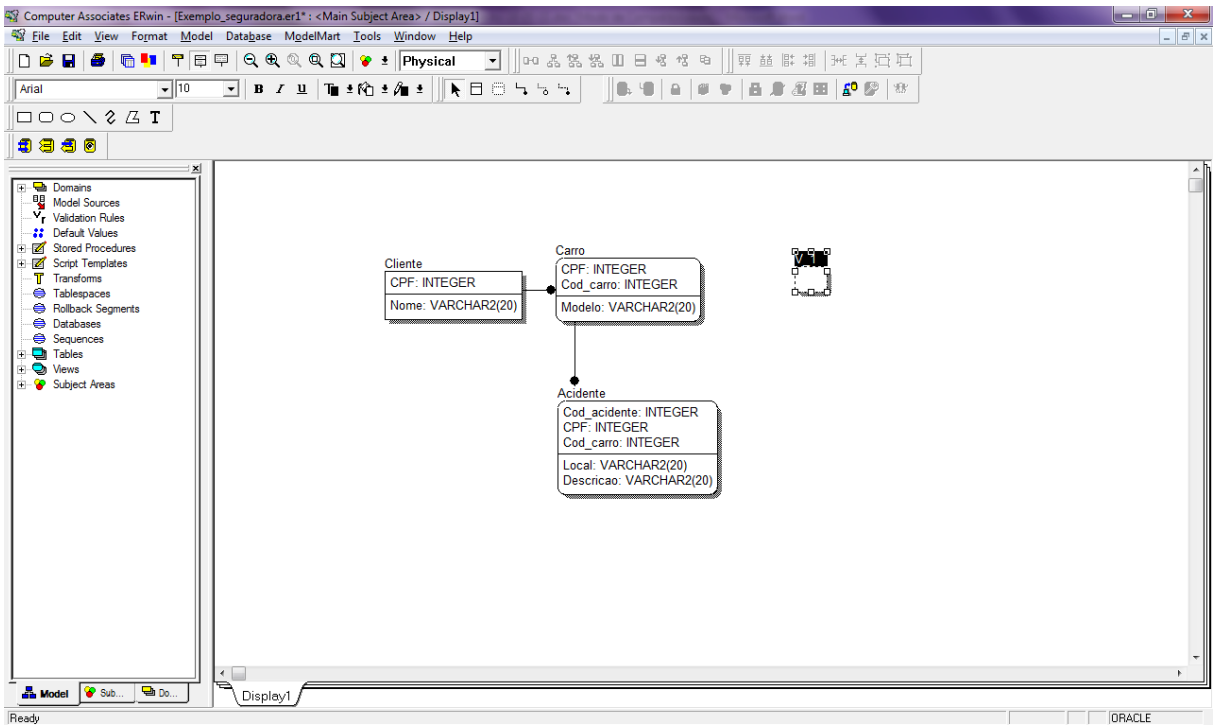


Figura 17. Interface do software ERwin.

3.3 Banco de dados – SQLite

Para armazenar os dados dos usuários bem como as ações dos mesmos e as situações-problema a serem solucionadas, possibilitando assim, a visualização do histórico das ajudas fornecidas e dos erros cometidos pelo usuário, foi utilizado o SQLite.

SQLite é um banco de dados relacional desenvolvido na linguagem C e que utiliza comandos SQL (*Structured Query Language*) - linguagem de pesquisa declarativa para banco de dados relacional baseadas na álgebra relacional.

Este banco de dados é bastante leve e muito utilizado para aplicativos de pequeno porte, suportando bases de dados acima de 2 *terabytes.* Umas das vantagens do SQLite é que para ser utilizado não é necessário fazer nenhum tipo de instalação, o banco de dados trata-se de um único arquivo que pode ser embutido no aplicativo. O SQLite não é uma biblioteca cliente usada para conectar com um grande servidor de banco de dados, mas sim o próprio servidor. A biblioteca SQLite lê e escreve diretamente para e do arquivo do banco de dados no disco (SQLITE, 2011).

Além disso, SQLite é multiplataforma e de domínio público, portanto, livre para ser utilizado com qualquer finalidade, comercial ou privada.



Figura 18. Logomarca do SQLite

4 Metodologia e Resultados

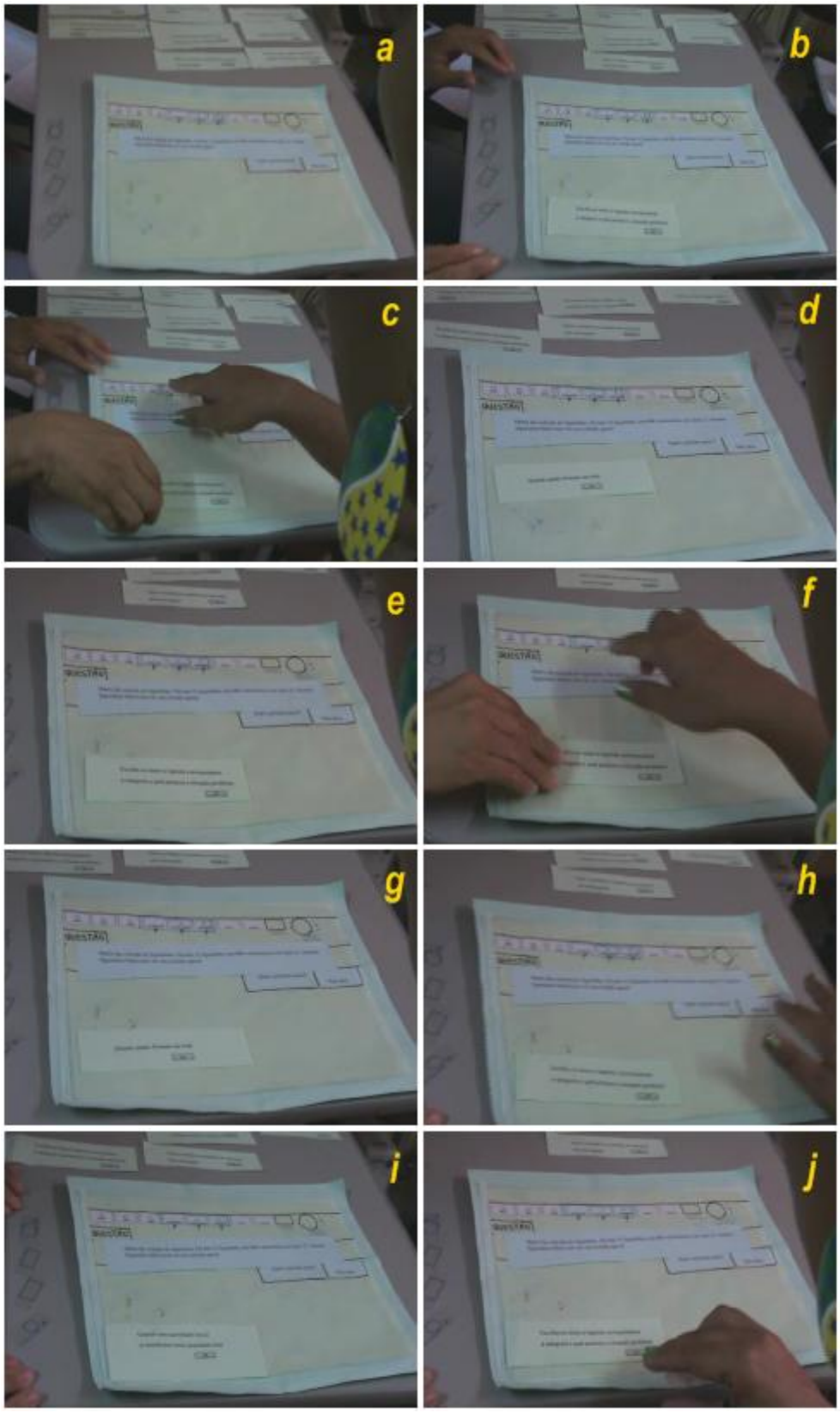
Os resultados aqui apresentados foram obtidos a partir de uma metodologia ágil de desenvolvimento. Segundo Abrahamsson et al. (2002), os métodos ágeis ou desenvolvimento ágil de software é um *framework* conceitual de engenharia de software, o qual promove um desenvolvimento iterativo através do ciclo de vida do projeto, minimizando os riscos, por meio de ciclos curtos, oferecendo, dessa forma, uma resposta mais rápida e flexível às demandas das empresas.

4.1 Identicação dos Requisitos

Para exemplificar situações de uso da interface educativa desenvolvida, apresentamos na Figura 19 um cenário de utilização da mesma, obtido a partir de um processo de prototipagem em papel. O processo de prototipagem em papel não integra as ações deste projeto. Ele foi realizado no âmbito de um projeto de iniciação científica e seus resultados foram transformados em requisitos, os quais representaram o ponto inicial para a construção da interface em linguagem de programação. Após os experimentos, nos quais, foi utilizada a técnica de prototipação e geração de animações, é apresentada uma análise dos quadros gerados nas animações. Tal análise favoreceu a identificação dos requisitos percebidos durante o experimento.

4.1.1 *Sotryboard* de utilização do GERARD

Storyboard significa esboço seqüencial de um filme ou um roteiro. A Figura 19 apresenta, através da prototipação, um *Stroyboard* de utilização da interface GERARD por um usurário. A seqüência de passos é assim descrita:


a
b
c
d
e
f
g
h
i
j

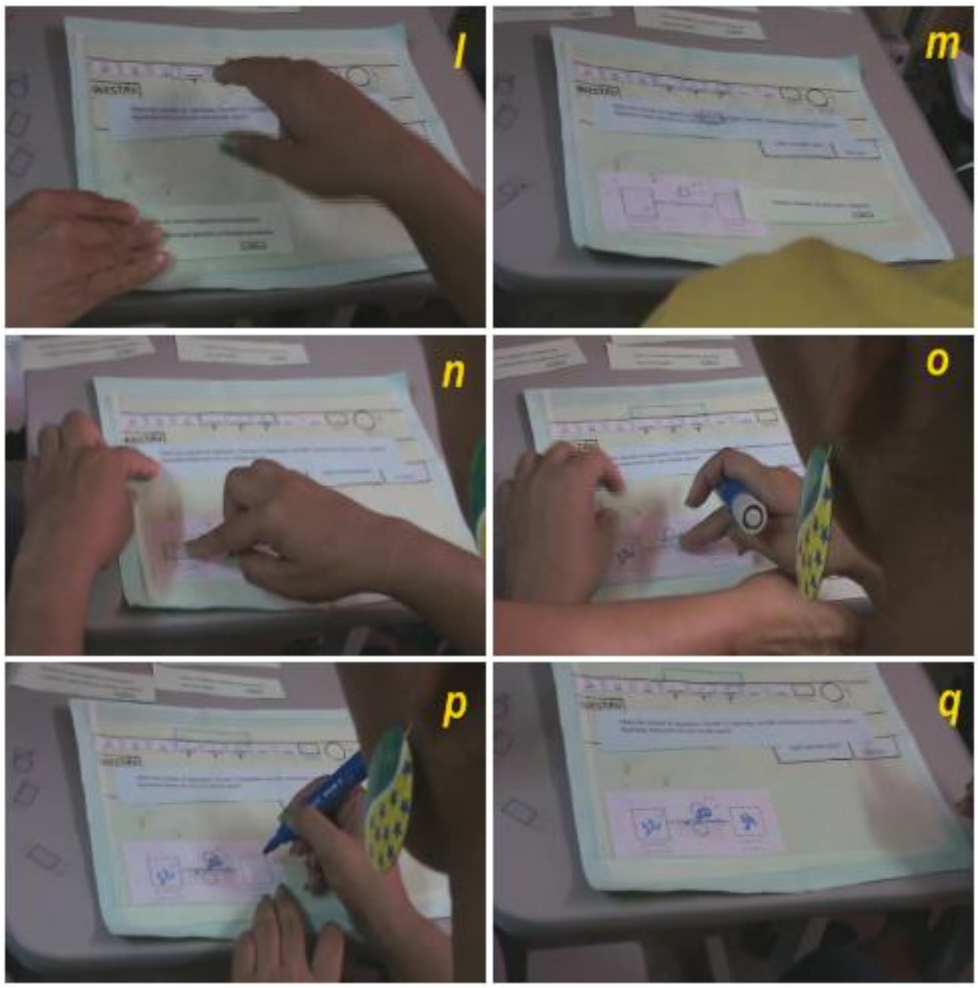


Figura 19. *Storyboard* da utilização da interface Gerard em prototipação.

- **Figura 19.a:** A interface pede para o usuário resolver a seguinte situação-problema: "*Maria faz coleção de figurinhas. Ela tem 32 figurinhas. Sua mãe a presenteou com mais 22. Quantas figurinhas Maria tem em sua coleção agora?"*.

- **Figura 19.b:** A interface exibe a seguinte mensagem para o usuário: *"Escolha no menu a legenda correspondente a categoria a qual pertence à situação-problema"*. Dessa forma, o usuário é convocado a descobrir de qual tipo de problema (Composição, Transformação, etc.) se trata a questão.

- **Figura 19.c:** O usuário “clica” no botão correspondente a legenda de Composição de Medidas, implicando assim no primeiro erro, que diz respeito à categorização do problema.

- **Figura 19.d:** Diante do erro cometido pelo usuário, a interface exibe a seguinte mensagem: *“Quando partes formam um todo”*. Esta mensagem é a definição de Composição de Medidas e reflete a ação anterior (“clicar” na legenda de Composição de Medidas) do usuário.

- **Figura 19.e:** Quando o usuário “clica” no botão “ok”, a caixa com a mensagem de ajuda desaparece e novamente aparece a caixa com a mensagem: *“Escolha no menu a legenda correspondente a categoria a qual pertence à situação-problema*”. Isto porque o usuário ainda não conseguiu descobrir se a situação problema se trata de uma Composição de Medidas, Transformação de Medidas, etc.

- **Figura 19.f:** Mais uma vez, o usuário “clica” no botão referente à Composição de Medidas, implicando assim no mesmo erro cometido anteriormente.

- **Figura 19.g:** A interface novamente exibe a mensagem “*Quando partes formam um todo”* uma vez que se trata do mesmo erro cometido pelo o usuário.

    - Essa mensagem não foi efetiva, ela é do tipo *Instruction*, um tipo de *feedback* que não foi efetivo nas situações de erro.
    - A mensagem foi substituída pelo tipo questionamento: “*Na situação-problema existem partes que formam um todo*?”.

- **Figura 19.h:** Ao “clicar” no botão “ok” da caixa que contém a mensagem “Quando partes formam um todo”, a interface novamente exibe a mensagem que pede para o usuário escolher a categoria a qual

pertence a situação-problema. No entanto, o usuário aperta o botão “mais dica” que oferece uma nova ajuda ao usuário.

- **Figura 19.i:** Ao “clicar” no botão “mais dica” a seguinte mensagem é exibida para o usuário: *“Quando uma quantidade inicial se transforma numa quantidade final*”. Esta mensagem tenta evidenciar para o usuário qual o tipo de problema que está sendo proposto.

    - Essa mensagem não foi efetiva, ela é do tipo *Instruction*, um tipo de *feedback* que não foi efetivo nas situações de erro.
    - A mensagem foi substituída pelo tipo questionamento: “*Na situação-problema uma quantidade inicial se transforma numa quantidade final*?”

- **Figura 19.j:** Quando o usuário “clica” no botão “ok”, a caixa com a nova mensagem de ajuda desaparece e novamente aparece a caixa com a mensagem: “*Escolha no menu a legenda correspondente a categoria a qual pertence à situação-problema*.”

- **Figura 19.l:** O usuário “clica” no botão referente a legenda de Transformação de Medidas, conseguindo assim descobrir a qual categoria pertence a situação-problema.

- **Figura 19.m:** A interface exibe o diagrama corresponde à Transformação de Medidas e a seguinte mensagem: “*Arraste os elementos do texto para o diagrama”*. Os elementos (que são os números 32 e 22 figurinhas) são evidenciados pela interface através das legendas referentes a número relativo e número natural (Seção 2.3).

- **Figura 19.n:** O usuário arrasta o número natural 32 para o lugar correspondente no diagrama.

- **Figura 19.o:** O usuário arrasta o número relativo 22 para o lugar correspondente no diagrama.

- **Figura 19.p:** Depois de ter montado o diagrama de forma correta, o usuário resolve o problema proposto.

- **Figura 19.q:** A interface apresenta o resultado final e parabeniza o usuário pela tarefa desempenhada com sucesso.

O processo descrito acima se trata da utilização da interface GERARD para ensino de estruturas aditivas. O mesmo processo de análise foi realizado para estruturas multiplicativas e pode ser visualizado no Anexo A.

4.1.2 Requisitos Verificados

Partindo dos trabalhos de Braga et al. (2008) e Queiroz et al. (2008), alguns requisitos foram identificados:

[REQUISITO_01] Requisito estático, que diz respeito ao que é visualizado em termos de botões, menus, área de desenho, etc.

A interface deve apresentar botões (ou itens de menu) que permitam a interação com o usuário. Deve apresentar também uma área que contenha a situação-problema a ser resolvida pelo usuário e uma área que possibilite a manipulação dos diagramas para resolução das situações-problemas

[REQUISITO_02] A interface deve apresentar *feedback* constante para o usuário sobre suas ações**.**

Sempre que o usuário interagir com o ambiente, este deve fornecer, através de mensagens, por exemplo, respostas que informem ao usuário o recebimento de um comando, tornando-o ciente da avaliação feita pelo *software* da ação que ele realizou.

[REQUISITO_03] A interface deve exibir através de uma janela de diálogo as instruções que indiquem para o usuário o que ele deve fazer.

O usuário necessita de informações que o situe e/ou norteie na utilização da interface para o desenvolvimento das situações-problemas. Isto é, ele precisa saber quais os passos a serem tomados e quais as ações esperadas pela interface.

[REQUISITO_04] As mensagens devem ser exibidas através de janelas de aviso, forçando o usuário a fechá-las primeiro para poder continuar a utilizar o aplicativo.

As mensagens possibilitam a interação dos usuários com a interface e são responsáveis por instigá-los à realização de ações que culminem ou não com a aquisição de algum conhecimento. Desta forma, as janelas que conterão as mensagens a serem exibidas devem paralisar a interface de forma que apenas mediante um clique no botão “ok” ela voltará ao normal.

[REQUISITO_05] A interface deve apresentar uma forma de ajuda que ficará disponível todo o tempo.

Esta ajuda será utilizada pelo usuário quando o mesmo não conseguir prosseguir no desenvolvimento de uma solução-problema.

[REQUISITO_05] A interface deverá evidenciar as partes a serem “arrastadas” para o diagrama no momento de compô-lo.

Depois de conseguir descobrir a qual categoria pertence a situação problema e, conseqüentemente, qual diagrama utilizar, o usuário é convocado a compor o diagrama com os valores descritos no problema proposto. A interface deve então evidenciar através das formas (quadrado ou círculo) as partes a serem arrastadas pelo o usuário até o diagrama.

[REQUISITO_06] A interface deverá fornecer um mecanismo para guardar (e abrir) o estado atual do usuário no desenvolvimento de um dado problema.

Por algum motivo, o usuário poderá querer finalizar o aplicativo e, em um momento posterior, querer utilizá-lo novamente a partir do ponto em que havia

parado. Dessa forma, a interface deve prover uma forma de salvar/abrir “estados” para os usurários.

[REQUISITO_07] A interface deve fornecer um mecanismo de refazer/desfazer ações realizadas pelo usuário.

Esse requisito está relacionado à necessidade do aluno de desfazer ações realizadas por engano com o software, bem como fornecer ao mesmo segurança para testar hipóteses e explorar possibilidades durante o aprendizado.

A identificação dos requisitos do GERARD foi realizada a partir da técnica de prototipagem em papel. Prototipação é uma técnica de construção de software na qual uma versão inicial de um sistema é disponibilizada rapidamente durante o desenvolvimento com o objetivo de elicitar e validar os requisitos da aplicação. (BRAGA, 2003 apud KOTONYA, 1998). O primeiro levantamento de requisitos do GERARD foi realizado por Braga (2006) e Queiroz e Gomes (2008). Com base nos requisitos e na prototipação em papel foi desenvolvido um protótipo da interface GERARD visualizada na Figura 20.

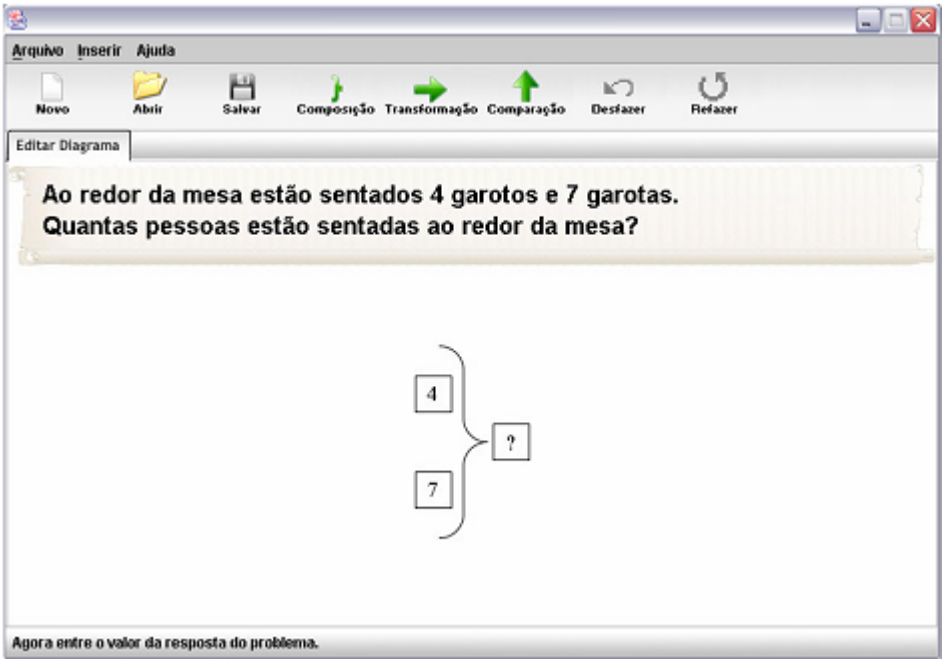


Figura 20. Interface do GERARD

De acordo com a Figura 20, a primeira versão do GERARD apresenta um menu com os artefatos: editor, novo, abrir, salvar e os elementos que

representam a estrutura aditiva. Além dessas, ainda estão presentes as operações de selecionar, refazer e desfazer. Na Tabela 1. Botões da barra de tarefas estão descritas, resumidamente, as funcionalidades dos botões da barra de tarefa.

Tabela 1. Botões da barra de tarefas

Botão	Funcionalidade
 Novo	Permite definir novos problemas, configurar as mensagens e a ajuda com material concreto. Não foi utilizado no teste com o usuário.
 Abrir	Permite carregar novos problemas para serem resolvidos. Não foi utilizado no teste com o usuário.
 Salvar	Grava os modelos criados pelo usuário para cada problema. Não foi utilizado no teste com o usuário.
 Composição	Insere na área de trabalho uma estrutura do tipo composição.
 Comparação	Insere na área de trabalho uma estrutura do tipo comparação.
 Transformação	Insere na área de trabalho uma estrutura do tipo Transformação.
 Selecionar	Seleciona um elemento do diagrama, permitindo alterar o seu valor.
 Refazer	Refaz a última ação realizada.
 Desfazer	Desfaz a última ação realizada.

Na resolução de problemas na primeira versão da interface GERARD, o primeiro passo ou ação é identificar a categoria à qual ele pertence. Isto é, se o problema se trata de uma Composição de Medida, Comparação de Medida ou

Transformação de Medidas. Feito isso, o processo de modelagem do problema pelo professor envolve o posicionamento dos números naturais ou relativos no modelo, seguidos da inserção do cálculo numérico. Uma representação dessa tarefa é apresentada na Figura 21. Tarefas realizadas durante a modelagem do problema..

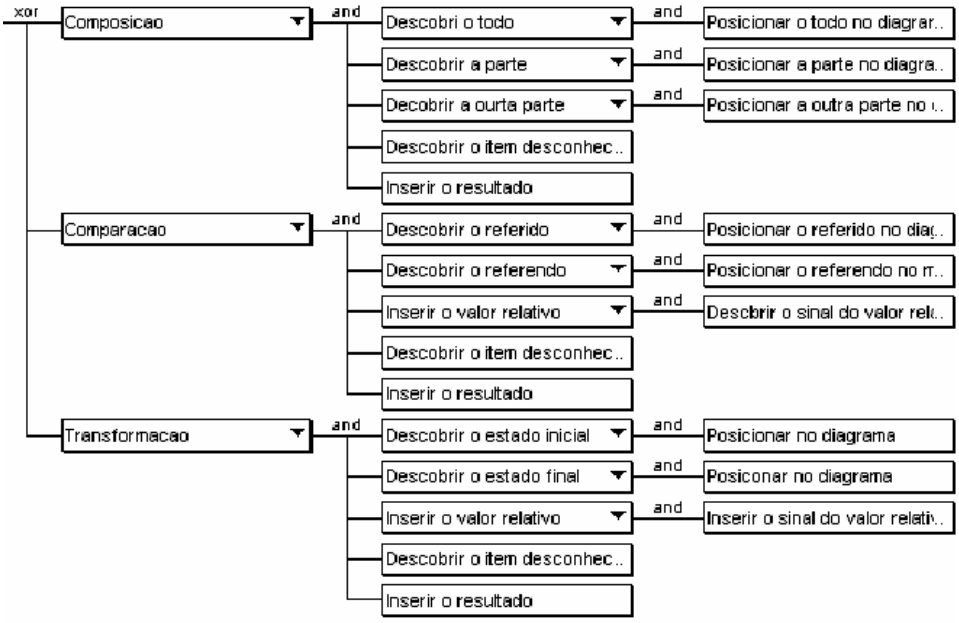


Figura 21. Tarefas realizadas durante a modelagem do problema.

4.2 Processo de modelagem

A modelagem possibilita a partilha de conhecimento entre diferentes tipos de pessoas. Para iniciar o processo de modelagem é necessária a utilização de um modelo. Este, por sua vez, é definido como a interpretação de um dado domínio do problema de acordo com uma determinada estrutura de conceitos. O modelo é especificado através de uma determinada linguagem (linguagem natural, UML, etc.) que pode ser textual ou gráfica. Quando esta representação é gráfica, nós a chamamos, usualmente, de diagrama (RAMOS, 2006).

4.2.1 Diagrama de Casos de Uso

Segundo Ramos (2006), os diagramas são conceitos que nos possibilitam agrupar elementos básicos e suas relações de uma forma lógica ou estrutural. Existem diferentes tipos de diagramas em UML. O diagrama de casos de uso é utilizado para descrever e definir os requisitos funcionais do sistema. Para isto ele contém alguns elementos gráficos como pode ser observado na Figura 22.

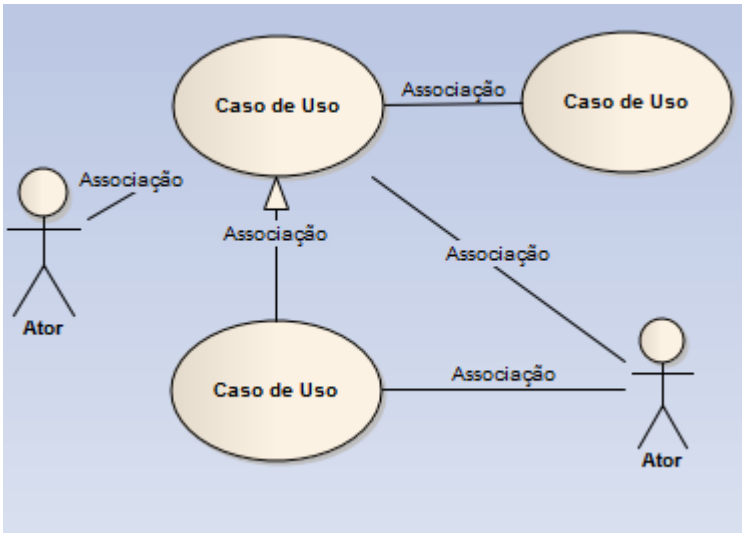


Figura 22. Modelo do Diagrama de Caso de Uso

- Caso de Uso: é a especificação da seqüência de ações que um ou mais atores realizam em um sistema de modo a obterem um resultado particular.
- Atores: entidades do sistema, pessoas ou outros subsistemas que interagem com o sistema em desenvolvimento.
- Associação: relacionamento entre casos de uso e/ou entre ator e caso de uso, indicando que há uma comunicação, possivelmente com envio e recepção de mensagens.

Os principais objetivos do diagrama de casos de uso são: descrever os requisitos funcionais do sistema, delimitar o sistema, oferecer as possíveis situações do mundo real para a fase de teste do sistema, servir de base para

usuários que não entendem de informática. (RUMBAUGH et al., 1999). O diagrama de casos de uso modelado para a interface GERARD é apresentado na Figura 23

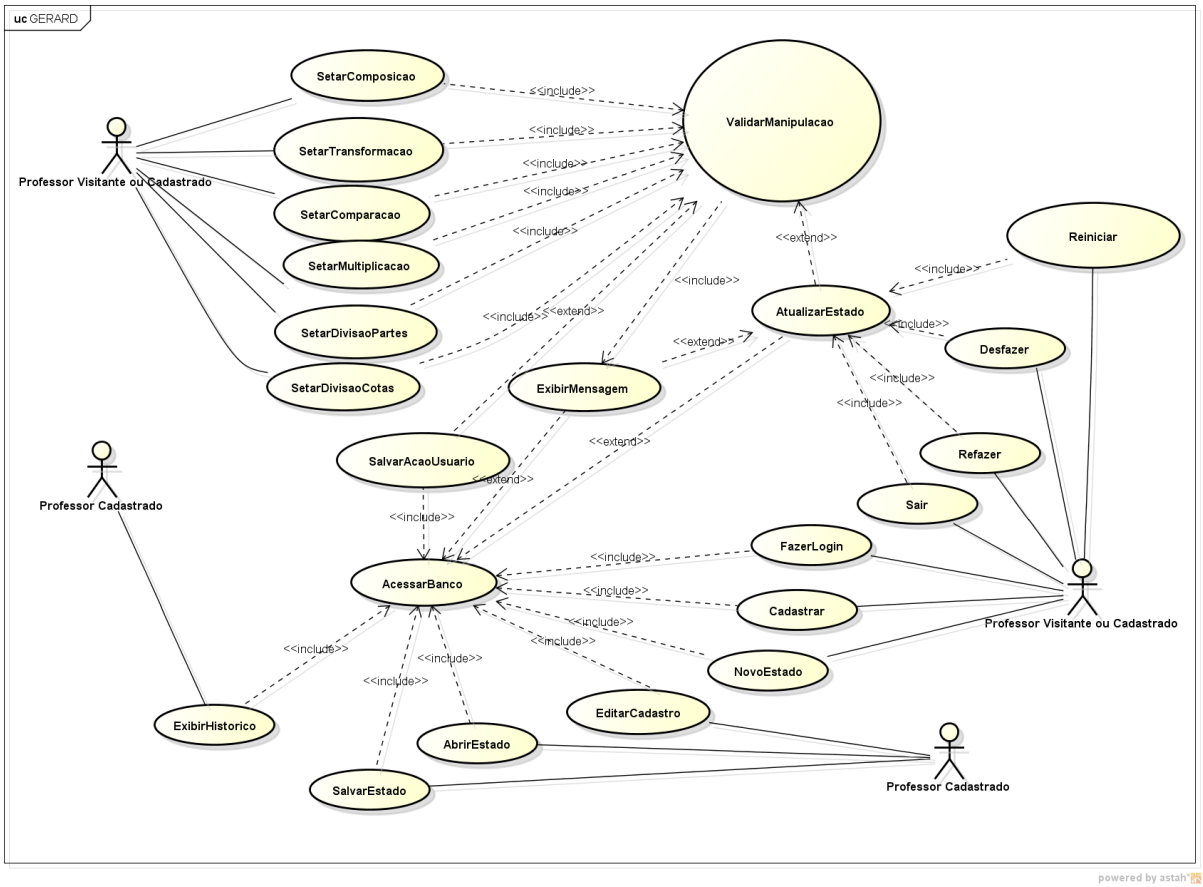


Figura 23. Diagrama de Casos de Uso do Gerard.

Cada caso de uso apresenta um cenário. Este corresponde a uma determinada seqüência de ações que ilustra um comportamento do sistema. (RAMOS, 2006). Numa abordagem de projeto com o envolvimento dos usuários, cenários podem ser criados para compor uma situação de estudo, servindo a antecipar uma situação futura e para testar alternativas. Os cenários relacionados ao diagrama de casos de uso apresentado na Figura 23 estão descritos no Anexo B.

4.2.2 Diagrama de classe

Diagrama de classe é o esquema mais comum encontrado em sistemas de modelagem orientado a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos (RUMBAUGH et al., 1999). Eles são utilizados para modelar a visão estática do sistema e têm grande importância não apenas para visualizar, especificar e documentar modelos estruturais, mas também para construir sistemas executáveis e muitas vezes poder aplicar engenharia reversa.

Assim como o diagrama de casos de uso o diagrama de classe é formado por elementos gráficos como observado na Figura 24. De acordo com a figura, tempos:

- Classe: é a representação abstrata de um conjunto de objetos. Como mostrando na Figura 24, a classe pode ser utilizada para representar os funcionário de uma escola, os professores, os dependentes dos funcionários, entre outros. A classe contém as características e comportamentos do objeto chamados, respectivamente, de atributos e métodos.
- Atributo: define as características da classe que o contém. Como mostrado na Figura 24, temos que uma característica do funcionário seria seu nome ou o seu salário. Além disso o atributo especifica o tipo de dado, isto é, se é um inteiro (número) ou uma string (texto) entre outros.
- Método ou Operação: corresponde ao comportamento de uma determinada classe, os métodos permitem a manipulação dos atributos ou obtenção de dados referentes a uma determinada classe como, no exemplo ilustrado na figura, a obtenção da grade de horários de um determinado professor.
- Associação: represente as relações existentes entre as classes. No diagrama apresentado na Figura 24, temos, por exemplo, um relacionamento de herança entre Funcionário e Professor, indicando que um Professor é um Funcionário. Entre Funcionário

e Dependentes temos uma associação de composição, indicando que o funcionário pode ter dependentes.

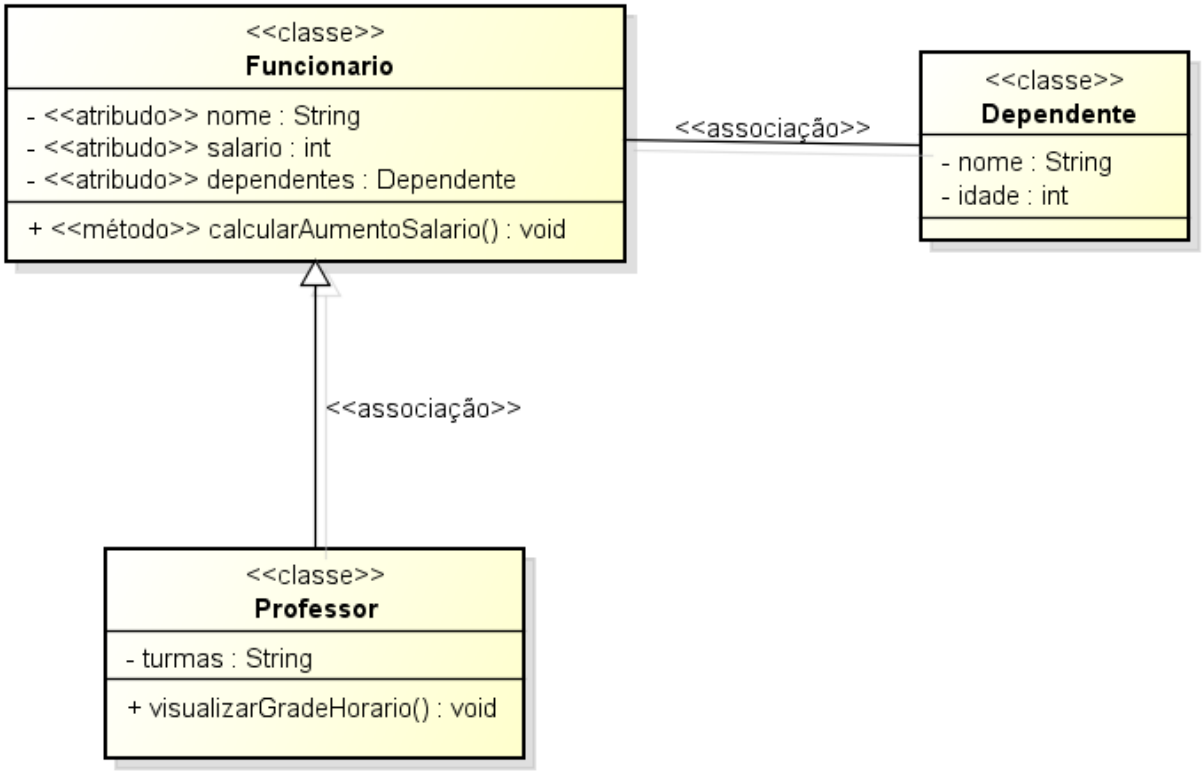


Figura 24. Modelo do Diagrama de Classe.

Os diagramas de classe modelados para desenvolvimento do software GERARD estão descritos no Anexo B, onde podem ser melhor visualizados.

4.2.3 Banco de dados

A modelagem do banco de dados foi feita a partir do trabalho realizado em (QUEIROZ, 2006), segundo a ontologia apresentada na Figura 26.

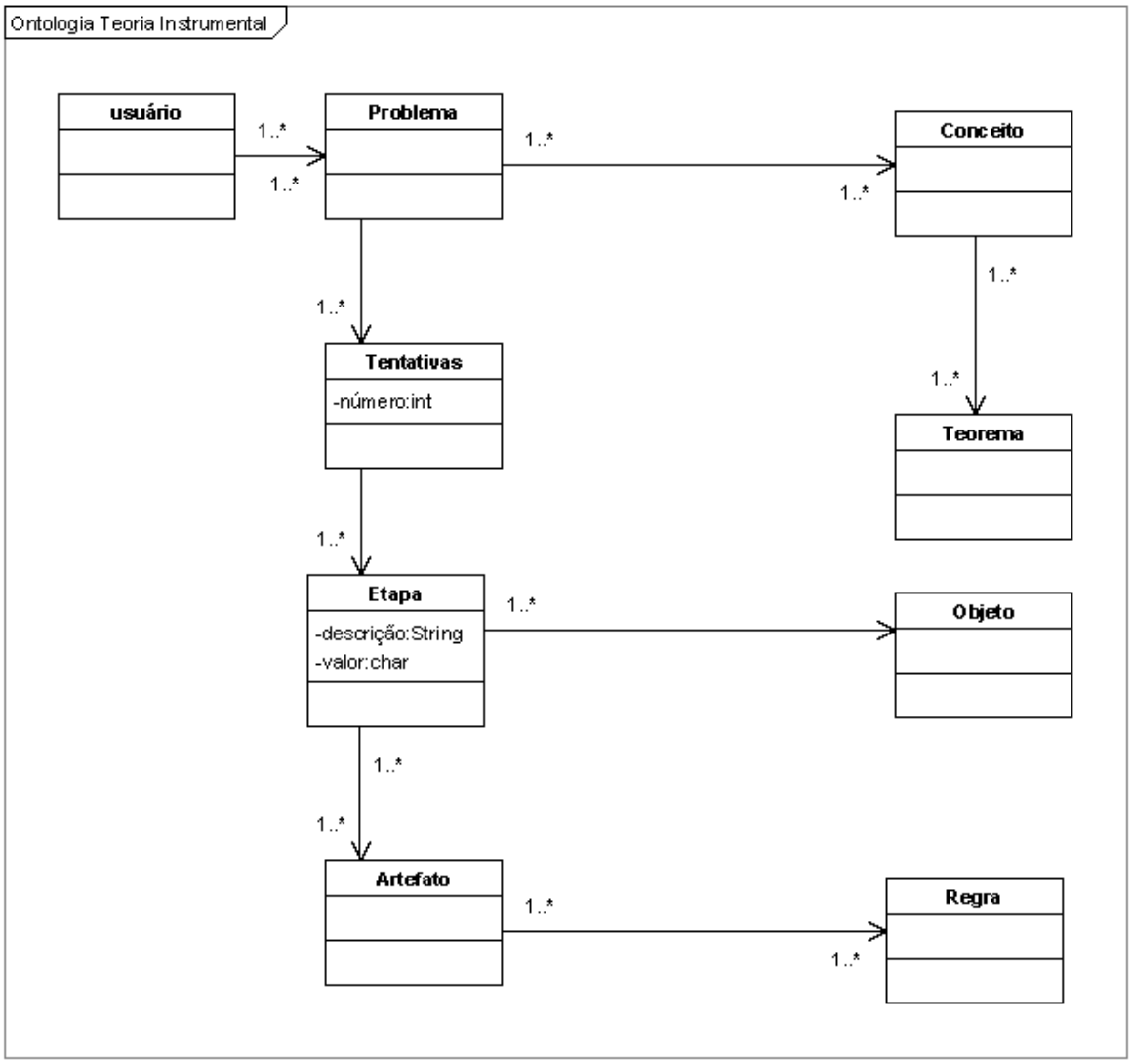


Figura 25. Ontotologida da teoria instrumental (QUEIROZ, 2006).

A modelo do banco utilizado pelo GERARD é descrito na Figura 26.

Este modelo de banco de dados favorece à realização da análise instrumental do software que, segundo Gomes (2005 apud QUEIROZ, 2006), é um modelo teórico utilizado para testes de usabilidade de interface, visando avaliar o potencial da mesma na aprendizagem de conceitos específicos durante sua utilização.

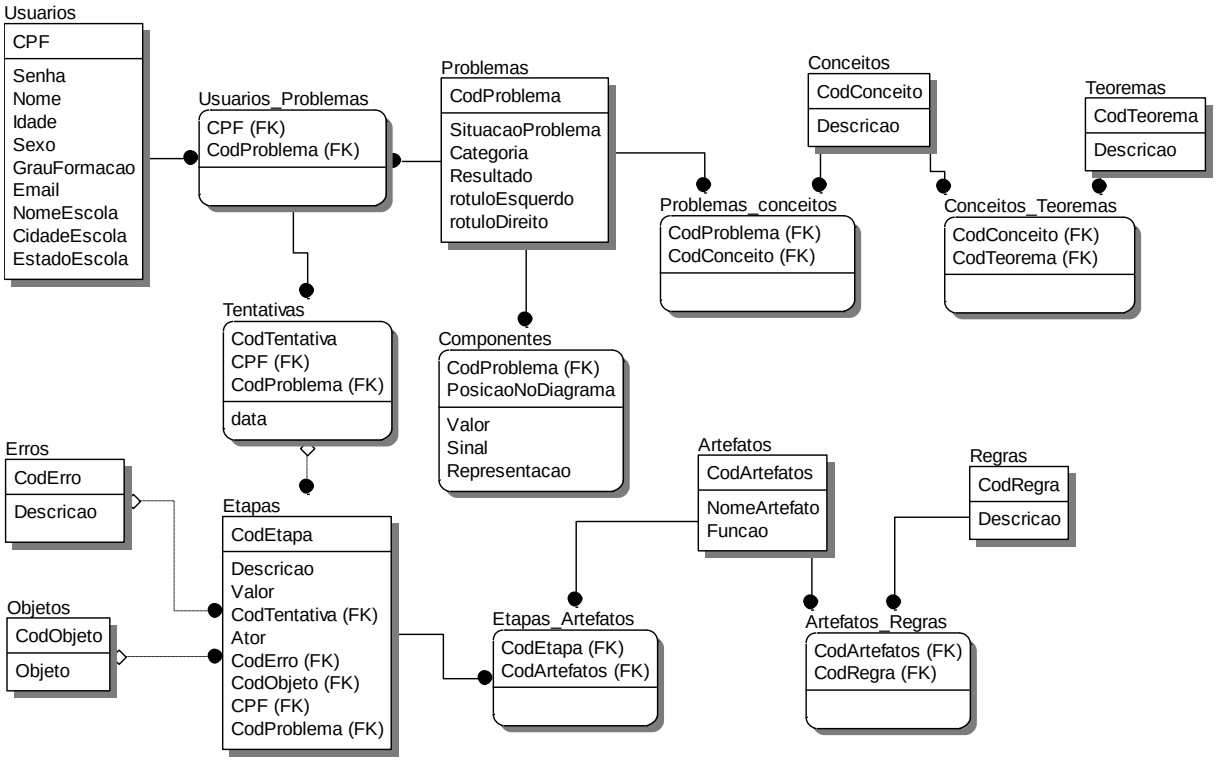


Figura 26. Modelo do banco de dados do software GERARD.

Segundo Queiroz (2006), os seguintes elementos podem ser observados para realização da análise:

- o usuário resolvendo o problema;
- o problema em resolução;
- os conceitos representados metaforicamente pela interface;
- Os teoremas-em-ato, mobilizados durante o uso da interface;
- Os artefatos utilizados em cada etapa de resolução do problema;
- Os objetos envolvidos na resolução do problema.

Na modelagem do banco foi acrescentada uma tabela para armazenar os erros cometidos pelo usuário durante a utilização do GERARD. De acordo com o levantamento de requisitos, estes erros podem ser especificados como apresentado na Tabela 2.

Tabela 2. Tipos de erros com respectivas mensagens exibidas para o usuário.

Tipo de erro		Mensagem exibida para o usuário
Erro categorização	Composição	Na situação-problema existem partes que formam um todo?
	Transformação	Na situação-problema existe um conjunto inicial que se transforma num conjunto final?
	Comparação	Na situação-problema existe um conjunto que está sendo comparado em relação a outro conjunto?
	Multiplicação	Na situação-problema o valor total é desconhecido?
	Divisão por partes	Na situação-problema o valor de cada parte é desconhecida?
	Divisão por cotas	Na situação-problema o número de partes é desconhecida?
posicionamento		Você tem certeza que o elemento foi posicionado corretamente? Clique "Ok" para tentar novamente!
sinal		Tem certeza que o sinal está correto? Clique "Ok" para tentar novamente!
calculo		Valor incorreto! Leia a questão novamente e digite o resultado!

4.2.4 Versão teste do GERARD e experimento

Como resultado do levantamento dos requisitos e da modelagem realizada, foi desenvolvida uma primeira versão (versão teste) do GERARD contemplando as categorias de estruturas aditivas com um número limitado de situações-problemas. A primeira versão pode ser observada na Figura 27.

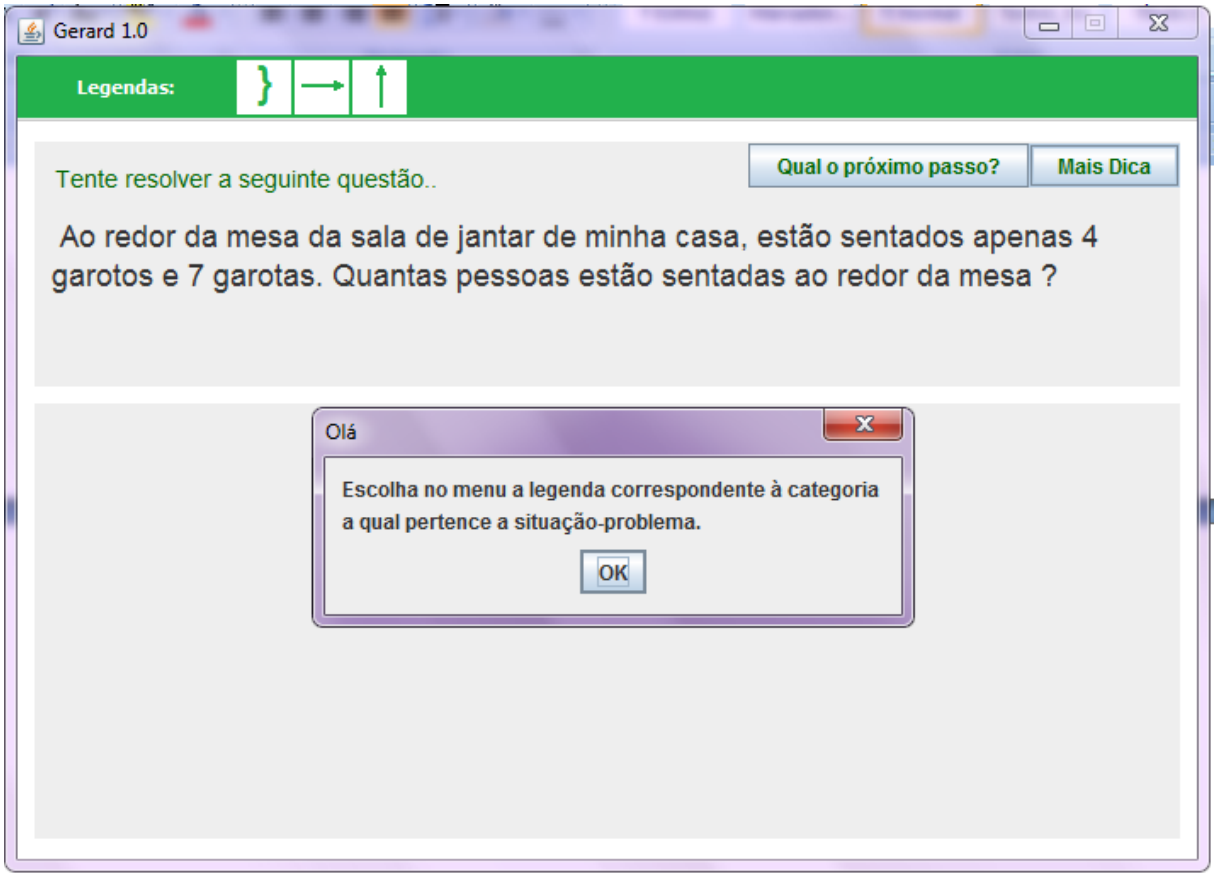


Figura 27. Versão teste do GERARD

Durante o desenvolvimento deste trabalho, foi possível realizar testes do aplicativo com os usuários (professores da escola M. Mª José Lima da Rocha – Juazeiro –BA) e validar os requisitos bem como verificar algumas contingências.

Os testes ocorreram em dois encontros (dois dias), contando com a participação de cinco professores. Os resultados obtidos estão sintetizados na Tabela 3.

Tabela 3. Resultados encontrados a partir dos testes realizados com os usuários.

Contingências verificadas	Adequação
A interface não solicita que o usuário atribua um sinal (mais ou menos) para o número relativo referente às categorias de Transformação e Comparação	Foi criada uma caixa de diálogo exibindo a mensagem: “Qual o sinal do número relativo?” e exibindo duas opções através do botão “mais” e botão “menos”.
Após posicionamento incorreto dos elementos do diagrama, a interface não exibe mensagem para o usuário, evidenciando o erro.	Após a identificação do posicionamento dos elementos no diagrama através da captura das coordenadas (x,y) do clique do mouse, foi criada a chamada para a exibição de mensagem evidenciando o erro para o usuário.
A interface não permite a maximização da tela, forçando o usuário a utilizar o software com um tamanho padrão e muitas vezes pequeno (dependendo da configuração do computador).	Foi feito o redimensionamento dos co
A interface apresenta falhas ao desenhar os diagramas, deixando os objetos em posições indesejadas	Foi introduzido o padrão de projeto *Decorator* para organizar a maneira como os desenhos são feitos e escolhidos dinamicamente na tela.

Depois de verificados os problemas levantados na Tabela 3, foi feita a remodelagem do software, empregando, quando necessário, padrões de projeto que agregaram confiabilidade e organização ao projeto.

5 O GERARD

5.1 Interface Gráfica com Usuário - GUI

Com base nos requisitos descritos no item 4.1 e nos resultados obtidos em Braga et al. (2008) e Queiroz et al. (2008), apresentamos o Gerard como segue na Figura 28, configuração para estruturas aditivas e Figura 29, configuração para estruturas multiplicativas.

Como pode ser observado nas Figura 28 e Figura 29, o GERARD apresenta um painel de botões que permitem a interação com usuário na resolução de situações problemas. As funcionalidades dos botões estão descritas na Tabela 4.

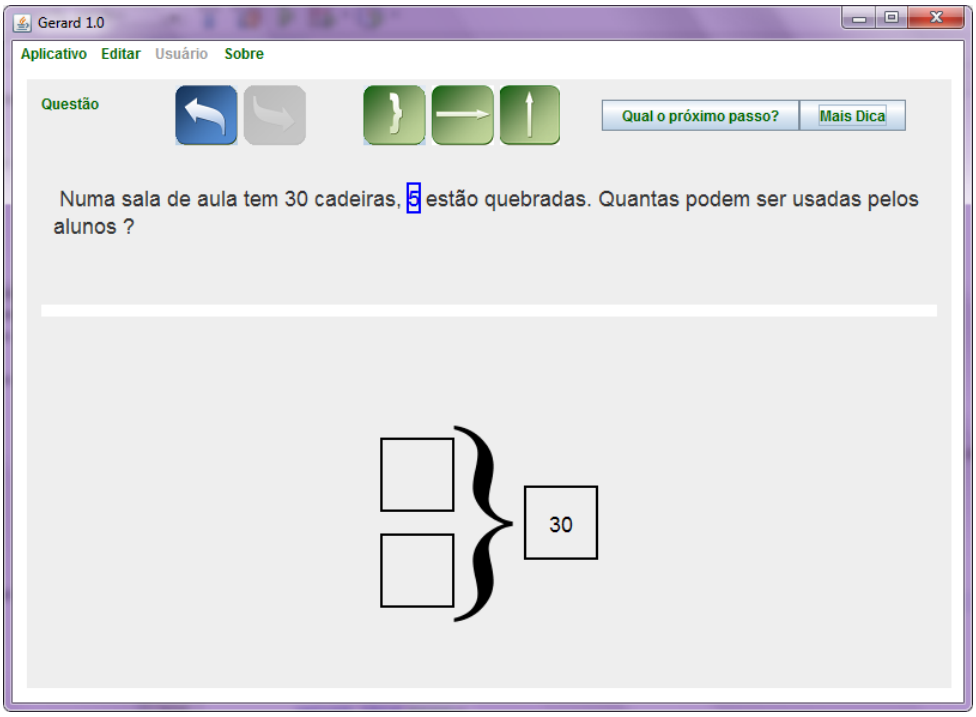


Figura 28. Interface do GERARD desenvolvido – estruturas aditivas

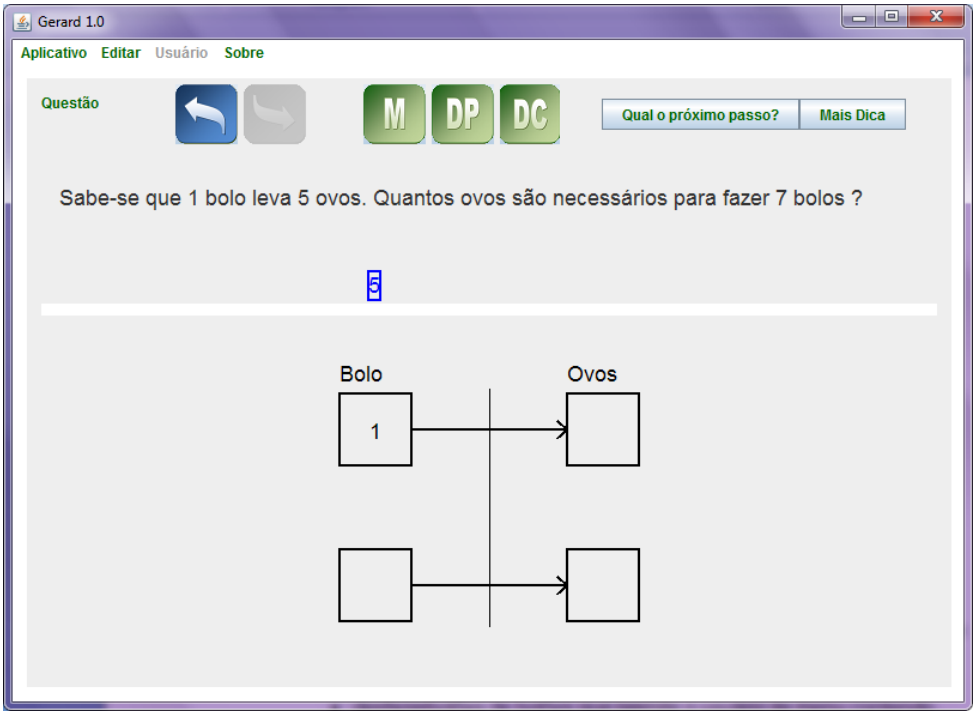


Figura 29. Interface GERARD desenvolvido - estruturas multiplicativas

Além do painel de botões, o GERARD apresenta uma barra de menus com os itens: “Aplicativo”, “Editar”, “Usuário” e “Sobre”. Estes, por sua vez, possuem os submenus mostrados na Figura 30.

Tabela 4. Descrição das funções do painel de botões.

Botões	Função
	Desfaz a ultima ação feita pelo usuário
	Refaz a última ação feita pelo usuário
	Exibe o diagrama de composição
	Exibe o diagrama de Comparação

	Exibe o diagrama de Transformação
	Exibe para o usuário, através de uma caixa de diálogo, o próximo passo que deve ser realizado no desenvolvimento da tarefa.
	Exibe para o usuário, através de uma caixa de diálogo, uma dica sobre para auxiliar no desenvolvimento da tarefa.
	Exibe o diagrama de Multiplicação
	Exibe o diagrama de Divisão por Partes.
	Exibe o diagrama de Transformação

Através do subitem “Estruturas”, é possível mudar a configuração do software, permitindo que o usuário escolha os tipos (multiplicativas ou aditivas) de situações-problema que ele deseja resolver.

O submenu “Usuário” pode ser visualizado apenas por usuários cadastrados. Além disso, apenas os usuários cadastrados têm acesso às opções “salvar” e “abrir” que têm as funções, respectivamente, de salvar a configuração atual na base de dados e abrir a ultima configuração salva pelo usuário. O

O submenu “Sobre” trás informações a respeito da interface GERARD.



Figura 30. Itens de menu do GERARD

Com a agregação da funcionalidade de exibição de histórico das ações dos usuários, foi necessária a criação da telas “login” e “cadastrado” apresentadas nas Figura 31e Figura 32.

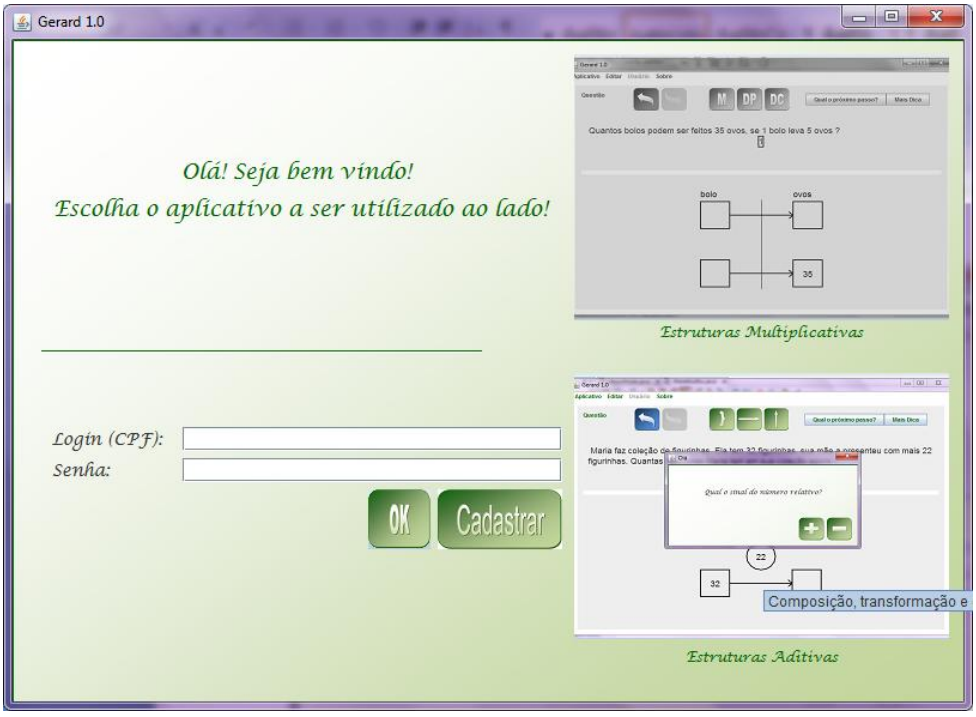


Figura 31. Tela login do GERARD.

Dessa forma, o usuário tem a opção de utilizar o software estando conectado ou usá-lo como visitante ou como usuário cadastrado.

Para os usuários que estão conectados, é possível visualizar o histórico das ações através do submenu “Exibir Histórico”. Caso haja dados na base, será exibida uma janela como apresentado na Figura 33.



Figura 32. Tela cadastro do GERARD.

O histórico do usuário apresenta, entre outros, a situação-problema resolvida, a data e para cada tentativa (ao resolver um determinado problema), são descritas as etapas realizadas pelo usuário. Como pode ser observado na Figura 33, cada etapa inclui os campos “Descrição”, “Ator”, “Valor” e “Erro”.

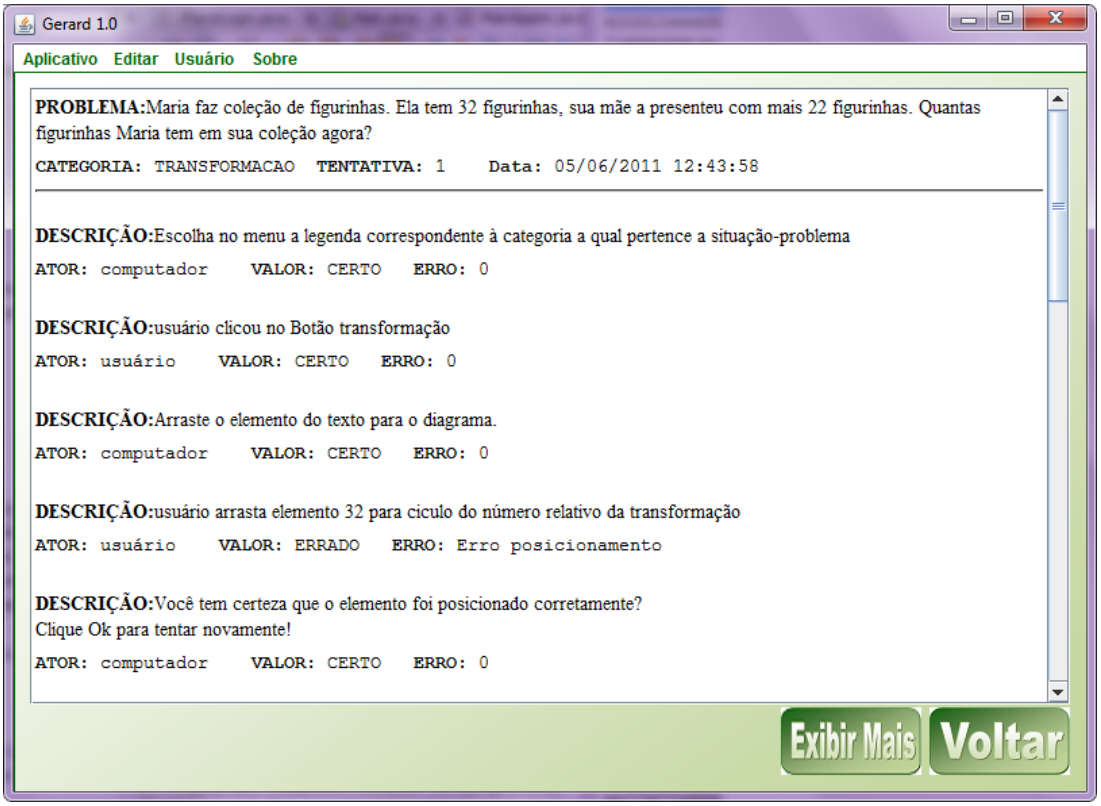


Figura 33. Tela de exibição de histórico do GERARD

As mensagem são apresentadas para o usuário através de uma janela de dialogo como exibida na Figura 34.

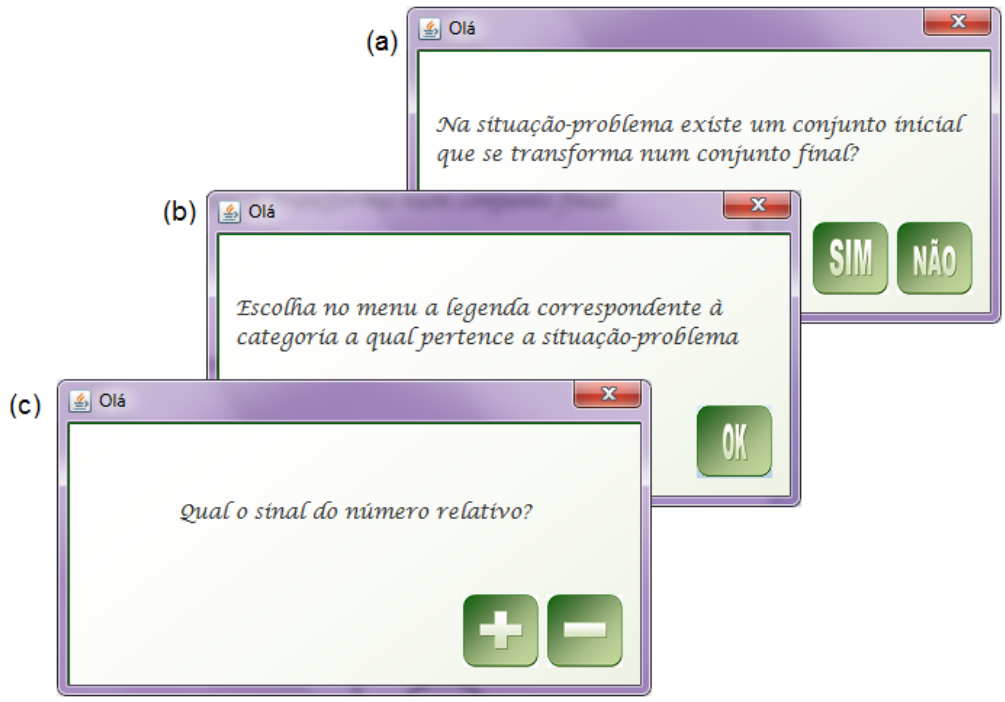


Figura 34. Janelas de diálogo do GERARD - (a) janela de questionamento. (b) Janela de instrução. (c) Janela de solicitação.

5.2 Arquitetura

O GERARD foi desenvolvido atentando para o os princípios de orientação a objetos (abstração, encapsulamento, modularidade e hierarquia) e utilizando-se padrões de projeto (*Desing Patterns*), agregando, assim, ao software os benefícios de OO e fazendo com que o mesmo seja aberto à extensões.

5.2.1 MVC – Model View Controller

Model-view-controller (MVC) é um padrão de arquitetura de software utilizado para separar a parte lógica da apresentação. Ele torna possível o

desenvolvimento, teste e a manutenção de cada uma isoladamente. De acordo com a arquitetura temos:

- **Model:** modela os dados e o comportamento por atrás do processo de negócios, se preocupando apenas com o armazenamento, manipulação e geração de dados
- **View:** inclui os elementos de exibição no cliente: HTML, XML, ASP, Applets, se trata da camada de interface com o usuário. Encarrega-se de receber e apresentar dados para o usuário.
- **Controller:** determina o fluxo da apresentação servindo como uma camada intermediária entre a camada de apresentação e a lógica.

Para desenvolvimento do GERARD foi aplicado o modelo MVC, contendo no total: um *controller*, dois *models* e seis *views*. O modelo MVC utilizado no GERARD pode ser visualizado nas Figura 35Figura 36.

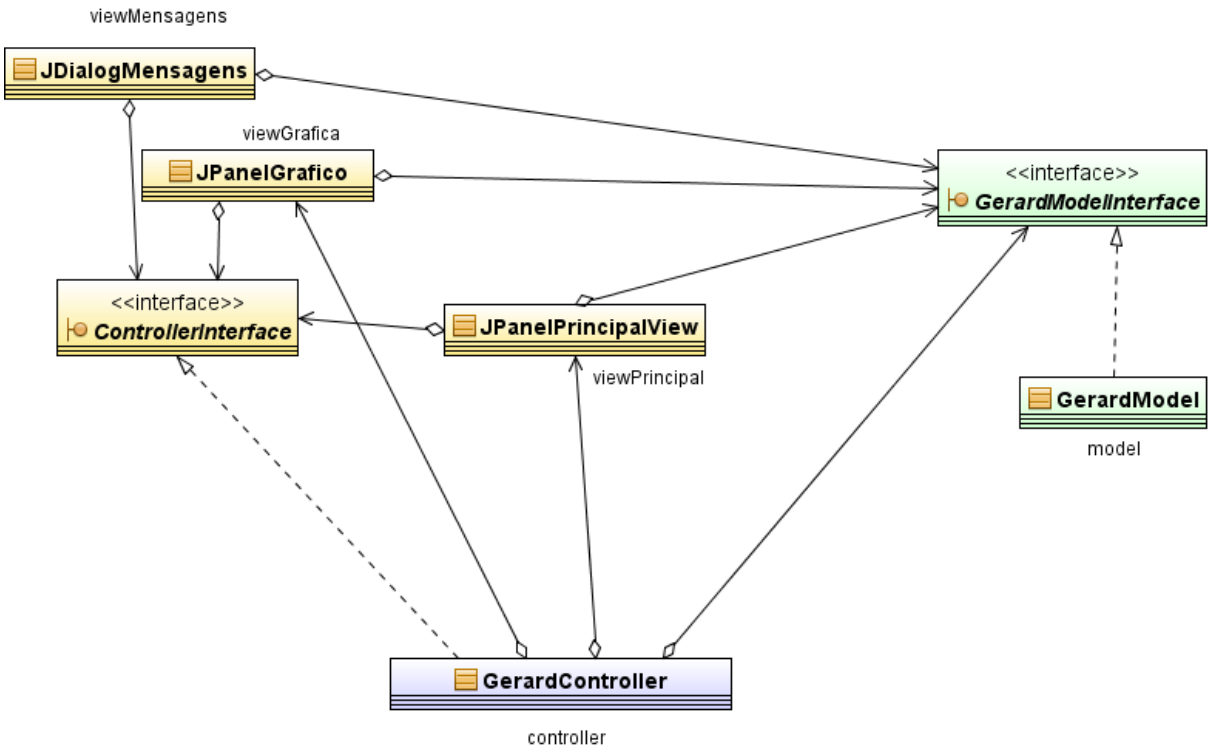


Figura 35. MVC1 – GERARD

De acordo com a Figura 35, o *model* “GerardModel” representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados, como por exemplo, verificar se o posicionamento do elemento no diagrama foi feito corretamente e, se o mesmo foi feito, quais são as atualizações que devem ser feitas. O *model*, como pode ser observado, não possui nenhuma instância do *controller* ou das *views* fazendo com que o mesmo seja independente. Em outras palavras, poderíamos trocar de interface e de controlador sem precisar fazer alterações no *GerardModel*.

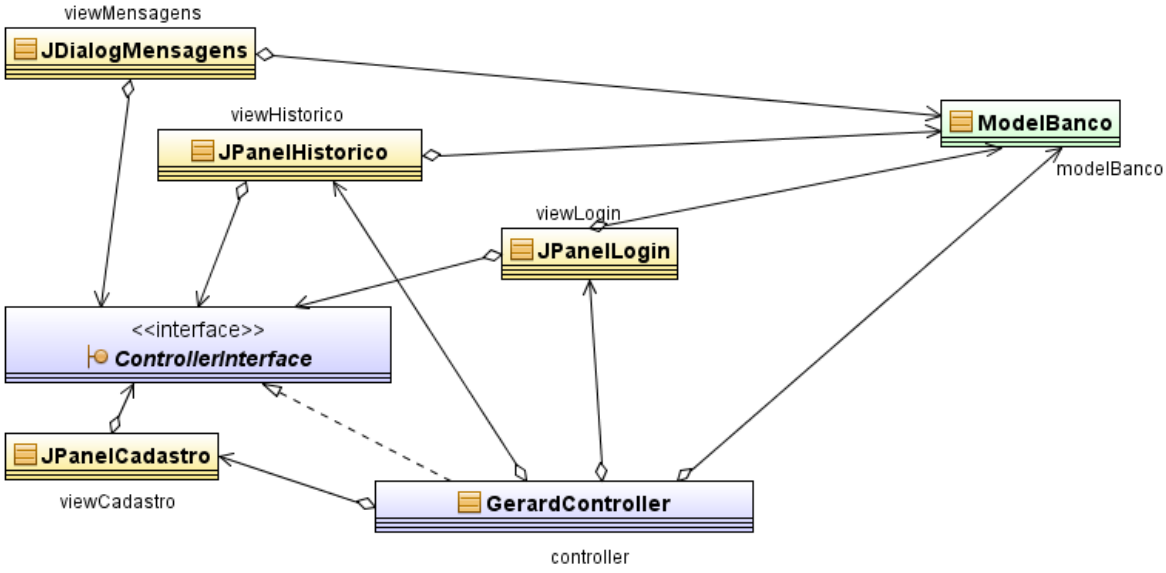


Figura 36. MVC2 - GERARD

As views "JDialogMensagens", "JPanelCadastro", "JPanelGrafico", "JPanelHistorico", "JPanelLogin" e "JPanelPrincipalView" (Figura 35Figura 36) são responsáveis pela apresentação dos dados para o usuário (através das GUIs) e pela entrada de dados oriundos dos mesmos. Como pode ser visualizado, as views possuem uma instância do *controller* (“ControllerInterface”) através da qual são enviadas mensagens quando um usuário interage com o software.

O controlador “GerardController” é responsável por enviar as mensagens das *views* e acionar o *model* para fazer os cálculos necessários.

Essa modelagem do sistema permite maior flexibilidade, manutenabilidade, e escalonabilidade. Isto é, viabiliza a agregação de funcionalidades e sistemas e, assim, conseqüente crescimento do software.

5.2.2 Padrão de projeto *Decorator*

O desenvolvimento do GERARD foi realizado de forma iterativa, fazendo com que a cada final de ciclo as etapas finalizadas fossem validadas ou modeladas novamente.

Depois da finalização da primeira versão do GERARD, foi inserido o padrão de projeto Decorator para fazer os desenhos gráficos e permitir maior organização do projeto. A modelagem realizada utilizando-se o padrão pode ser observada na Figura 37.

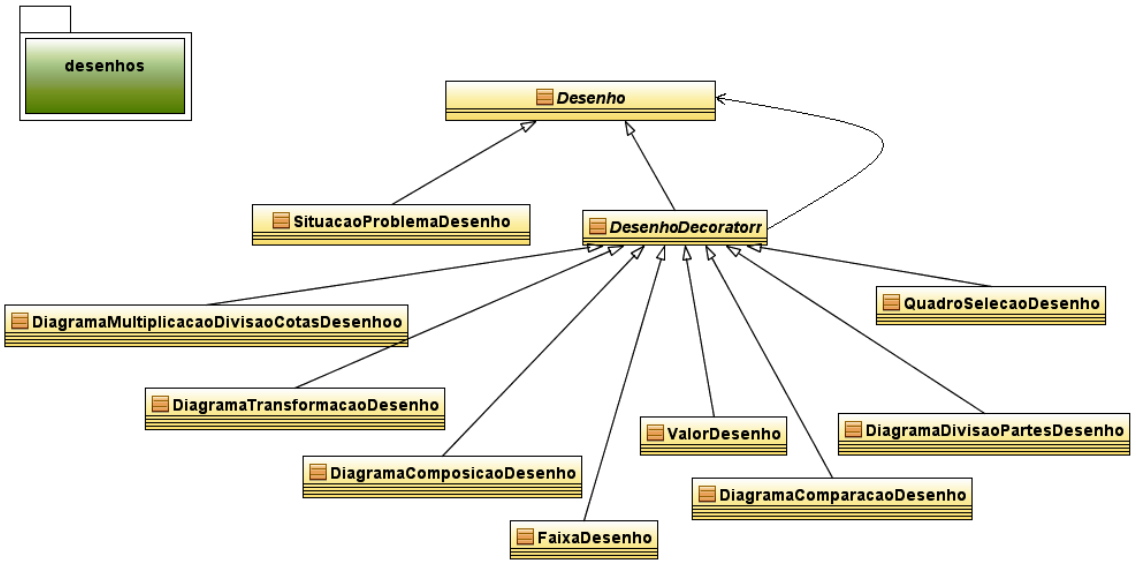


Figura 37. Padrão de projeto Decorator - GERARD

O Decorator é utilizado quando se deseja adicionar responsabilidades para um objeto dinamicamente (FREEMAN et al., 2004). No caso do GERARD,

estas responsabilidades correspondem aos desenhos que podem vir ou não a serem renderizados pela interface, uma vez que as mesmas dependem das ações dos usuários.

Dessa forma, quando o GERARD é inicializado, primeiramente, é feito o desenho textual da situação-problema através da classe “Desenho” e a herança com a classe “SituacaoProblemaDesenho”. Com a utilização do software pelos usuários, são (ou não) agregados (decorados) outros desenhos à tela. Isso é feito através da classe “DesenhoDecorator” e a herança com as demais classes, como mostra a Figura 37.

O padrão de projeto *Decorator* permite mudar o comportamento dos componentes através da adição de novas funcionalidades a partir da herança e do polimorfismo, atentando para o princípio *Open-Closed* de OO (orientação a objetos), segundo o qual, o software deve ser fechado para modificação e aberto para extensão (FREEMAN et al., 2004).

Sendo assim, é possível expandir o GERARD para abranger um range maior de categorias de situações problemas, apenas criando novas classes (para fazer os desenhos dos diagramas) sem precisar modificar o código já desenvolvido

5.2.3 Padrão de projeto *State*

Como pôde ser observado nas seções anteriores, o GERARD possui quatro tipos de configurações ou estados:

- Aditivas: apresentando a lógica para tratar as ações relacionadas às categorias de Composição, Transformação e Comparação;
- Aditivas para usuários cadastrados: apresentando as mesmas categorias das Aditivas, porém sendo necessário inserir as ações dos usuários no banco de dados;

- Multiplicativas: apresentando lógica para tratar as ações relacionadas às categorias de Multiplicação, Divisão por Partes e Divisão por Cotas;
- Multiplicativas para usuários cadastrados: apresentado as mesmas categorias das Multiplicativas, porém, sendo necessário inserir as ações dos usuários no banco de dados.

Dessa forma, podemos concluir que para cada configuração (escolhida pelo usuário) haverá uma lógica diferente a ser realizada pelo sistema. Geralmente em programação, situações como estas são tratadas com cláusulas “if e else”, que resultam em códigos muito extensos, não-confiáveis e difíceis de serem mantidos (FREEMAN et al., 2004).

Para contornar este problema no GERARD, foi utilizado o padrão de projeto *State*, o qual permite que um objeto altere seu comportamento se seu estado interno for alterado, parecendo como se o próprio objeto tivesse alterado sua classe (FREEMAN et al., 2004).

O padrão State utilizado no GERARD pode ser visualizado Figura 38.

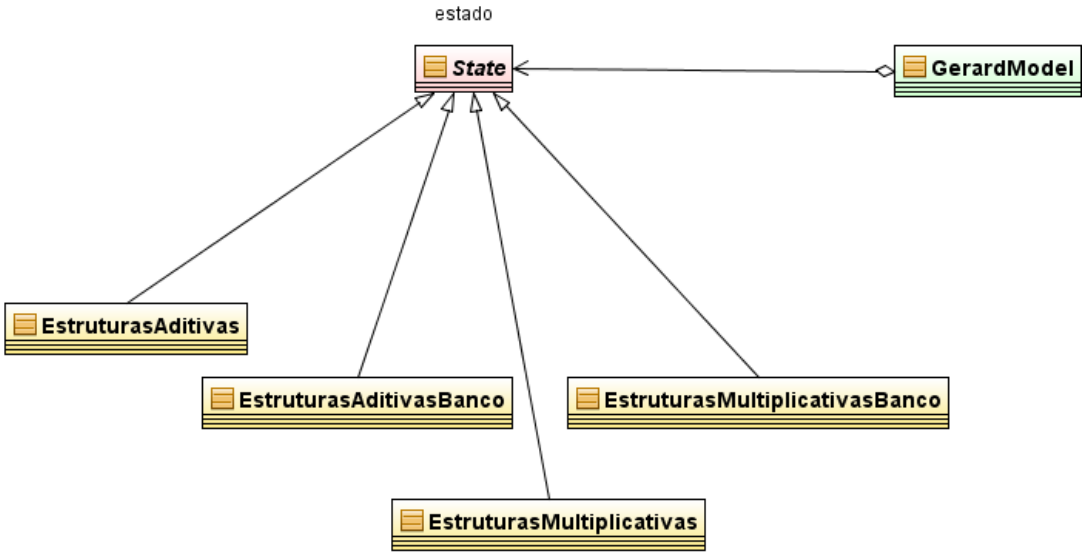


Figura 38. Padrão State - GERARD.

Como pode ser observado na Figura 38 e de acordo com a descrição do padrão State, a configuração e a lógica tratada para cada uma das quatro

configurações anteriormente descritas, é delegada às subclasses que herdam a classe State, sendo o processo, assim, transparente para o “GeradModel”.

O uso do padrão *State* faz com que o comportamento do estado seja localizado, porém, que o comportamento para diferentes subestado seja separado. Isto facilita qualquer modificação do comportamento do estado e, em particular, a adição de estados extra.

6 Conclusão

O objetivo desse projeto foi desenvolver uma interface educativa atendendo os requisitos levantados através da prototipagem em papel que foi realizada no âmbito de um projeto de iniciação científica. Para isso, foram aplicados conhecimentos adquiridos durante a formação, sobretudo de engenharia de software.

Foram utilizados padrões de projetos consolidados para organizar e arquiteturar o software, agregando ao mesmo características como escalabilidade, confiabilidade e manutentabilidade.

6.1 Contribuíções

O software aqui desenvolvido será utilizado na Escola M. Mª José Lima da Rocha localizada na cidade de Juazeiro-BA como parte do projeto de formação continuada de professores.

Este trabalho de conclusão de curso não se trata apenas de um desenvolvimento de um software, mas, deste (desenvolvimento de software), atrelado à aplicabilidade prática e à contribuição dos conhecimentos adquiridos durante a formação do curso de engenharia da computação, para melhorar a educação, no âmbito do ensino-aprendizagem da Matemática.

Parafraseando Gérard Vernaud, “*é primordial, ainda que seja necessário ter consciência de que não existem milagres, que ninguém vai conseguir eliminar todos os problemas de um dia para o outro. Mas, se podemos dar ao professor os meios de conhecer melhor seu trabalho, os limites de sua ação, os obstáculos que vão encontrar e as formas de controlar a evolução das turmas, é absurdo não fazer isso”*.

6.2 Trabalhos futuros

O GERARD foi desenvolvido tanto para ser usado como um applet quanto como um aplicativo para descktop. Estas possibilidades permitem que o GERARD seja utilizado para educação à distância, sendo necessária a criação de um servidor e a implementação de servlets para busca de informação na base do servidor e comunicação com o software.

7 Referências

ABRAHAMSSON, P. et al. **Agile software development methods: review and analysis.** Finlândia: VTT Publications, 2002. 107 p.

ALVARES, A. J. ; FERREIRA, J. C. E. ; SHIMABUKURO, L.; FERNANDES, C. A. **Um sistema de telemanufatura baseado na web orientado ao processo de oxicorte**. In: XXII Encontro Nacional de Engenharia de Produção, ENEGEP 2002, 2002, Curitiba. XXII Encontro Nacional de Engenharia de Produção, ENEGEP 2002, 2002

ASTAH, **astah* community - FREE UML Modeling Tool.** Disponível em: http://astah.change-vision.com/en/. Acesso em: 15 Mai. 2011

.

BRAGA, M. M. **Design de software educacional baseado na Teoria dos Campos Conceituais**. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Pernambuco, 2006.

DEITEL, H. M.; DEITEL, P. J. **Java: como programar.** Editora Bookman. 6ª ed. São Paulo: 2005.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 4ª ed. Editora Addison-Wesley. 2005.

FISCHER, M. C. B. O. **Estudo de Requisitos para um Software Educativo de Apoio ao Ensino da Introdução à Computação**. Dissertação de Mestrado. Local: USP – Universidade de São Paulo, 2001

FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES **Head First Design Patterns.** O'Reilly Media, Inc., 1 edition, October 2004.

GOMES, A. S.; CASTRO FILHO, J. A. de; BARRETO, M. **Competências matemáticas de alunos de primeiro e segundo ciclos em situações aditivas e multiplicativas.** In: VIII Encontro Nacional de Educação Matemática, 2004, Recife. Anais do VIII Encontro Nacional de Educação Matemática. Recife: SBEM-PE, 2004

GOMES, A. S.; QUEIROZ, A. E. M.; BRAGA, M. M. **Design de Software Educacional Baseado na Teoria dos Campos Conceituais**. In: XIX Simpósio Brasileiro de Informática na Educação (SBIE), 2008, Fortaleza.

GOMES, A. S.; WANDERLEY, E. G. . **Elicitando requisitos em projetos de Software Educativo**. In: WIE 2003 Workshop Brasileiro de Informática Educativa, 2003, Campinas. Ciência, Tecnologia e Atalhos para o futuro - Anais do XXIII Congresso da Sociedade Brasileira de Computação. Campinas: SBC, 2003. V. p. 227-238

GRAVINA, M. A; SANTAROSA, L. M. **A Aprendizagem da Matemática em Ambientes Informatizados**. In anais do Congresso Ibero-Americano de Informática na Educação. 4, 1998, Brasília

INEP, inep.gov. **Nível de leitura e matemática da maioria dos alunos é "crítico".** Disponível: http://vvovw.inep.gov.br/imprensa/noticias/saeb/news03_02.htm. Acesso em: 25 Nov. 2010.

MAGINA, S., CAMPOS, T., NUNES, T. E GITIRANA, V. **Repensando a Adição e a Subtração: contribuições da Teoria dos Campos Conceituais**, São Paulo, 2000. PROEM-PUC/SP.

MISKULIN, R., PEREZ, G., SILVA, M., MONTREZOR, C., SANTOS, C., TOON, E, LIBONI FILHO, P., SANTANA, P.. Identificação e Análise das Dimensões que Permeiam a Utilização das Tecnologias de Informação e Comunicação nas Aulas de Matemática no Contexto da Formação de Professores. **Bolema: Mathematics Education Bulletin = Bolema: Boletim de Educação Matemática**, América do Norte, 19, out. 2008. Disponível em: http://www.periodicos.rc.biblioteca.unesp.br/index.php/bolema/article/view/1869/1630. Acesso em: 05 Nov. 2010.

NASCIMENTO, M. R. R., NASCIMENTO, A. A do. As tecnologias da informação e da comunicação na escola: uma questão de competência e desafio educacional. **Revista da FA7**, v. 2, n. 7 p. 51-60, 2009.

NETBEANS, NetBeans.org. Disponível em: http://www.netbeans.org. Acesso em: 12 Nov. 2010.

ORACLE. **Java.com**. Disponível em: http://www.oracle.com/technetwork/java/index.html Acesso em: 26 Nov. 2010.

PIRES, 2003 Relações entre aprendizagem e desenvolvimento: A abordagem de Jerome Bruner, Patrícia Vasconcellos Pires Ferreira. Disponível em http://www.caleidoscopio.psc.br/ideias/bruner.html. Acesso em: 01 Nov. 2010.

QUEIROZ A. E. M., **Design interativo de agentes inteligentes de interface para software educativo de Matemática**. Dissertação

(mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2006.

QUEIROZ A. E. M., GOMES, A. S. **Efeitos da manipulação direta reflexiva na resolução de problemas sobre estruturas aditivas**. Revista Brasileira de Informática na Educação nrº. 2. Vol. 16. Agosto de 2008

RAMOS, R. A. **Treinamento Prático em UML**. 1. ed. São Paulo: Digerati, 2006. v. 1. 144 p

RUMBAUGH, J., JACOBSON, I., BOOCH, G. **The Unified Modeling Language Reference Manual**, Addison-Wesley, Reading, MA, 1999, ISBN 0-201-30998-X

SANTOS, J. G. Conceitos de Banco de Dados. Disponível em: http://pusivus.com.br/apostilasdoc/ConceitosBancoDados.htm Acesso em: 15 Mai. 2011.

SQLITE, **About SQLite**. Disponível em: http://www.sqlite.org/books.html. Acesso em: 01 Jun. 2011

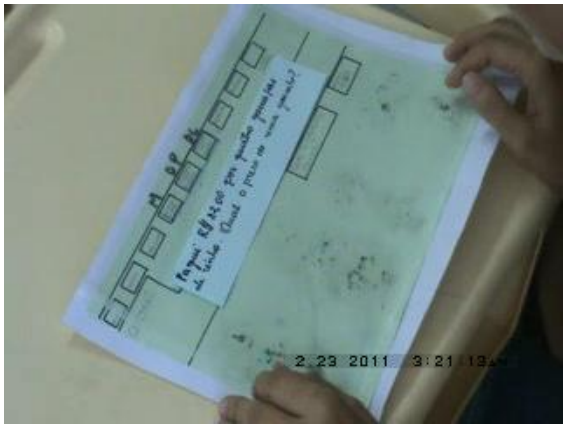
VERGNAUD, G. **Multiplicative structures**. In J. Hiebert & M. Behr (Eds.) Number Concepts and Operations in the Middle Grades (pp. 141-161), 1988. Reston: NCTM.

Anexo A

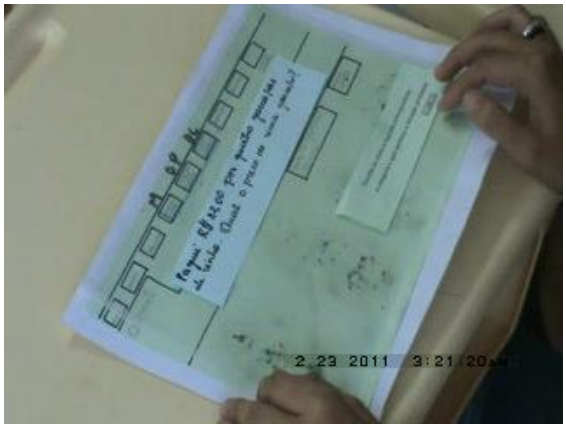
Análise através do *storyboard* para o uso do GERARD para Estruturas Multiplicativas

Sotryboard construido a partir da utilização do GERARD através da prototipação em papel.

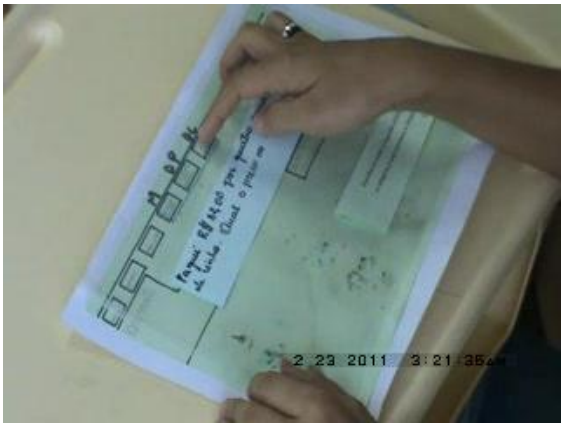
Análise para uso de estruturas aditivas, contemplando as categorias de Multiplicação, Divisão por Partes e Divisão por Cotas.



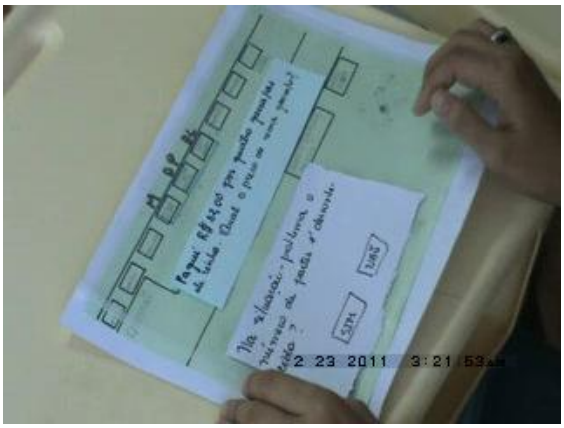
1. A interface pede para o usuário resolver a seguinte situação-problema: *"Pague R$ 12,00 por quatro garrafas de vinho. Qual o preço de uma garrafa?"*.



2. A interface exibe a seguinte mensagem para o usuário: *"Escolha no menu a legenda correspondente a categoria a qual pertence à situação-problema"*. Dessa forma, o usuário é convocado a descobrir de qual tipo de problema (Multiplicação, Divisão por partes ou Divisão por Cotas).



3. O usuário "clica" no botão correspondente a legenda de Divisão por Cotas, implicando assim no primeiro erro, que diz respeito à categorização do problema.

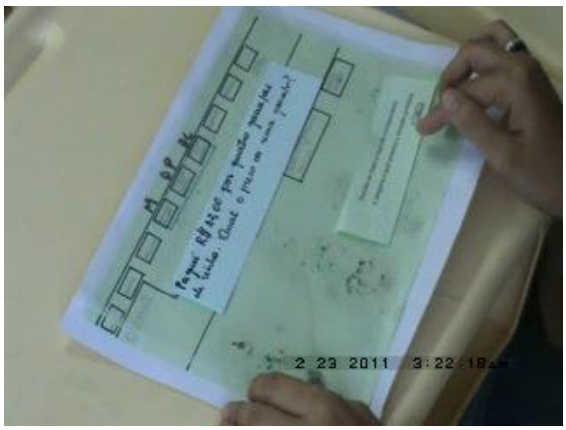


4. Diante do erro cometido pelo usuário, a interface exibe a seguinte mensagem: *"Na situação problema o número de partes é desconhecida?"*.

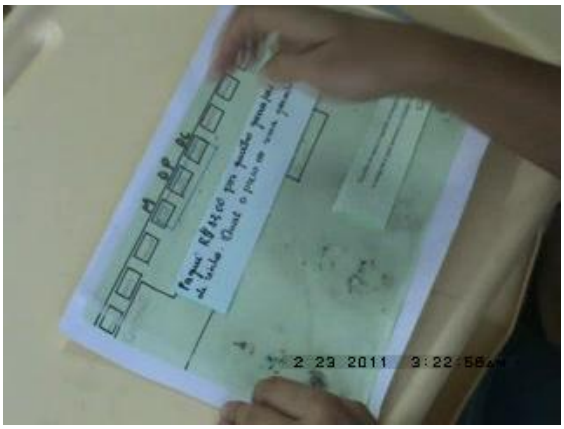
Esta mensagem é a definição de Divisão por cotas e reflete a ação anterior (“clicar” na legenda de divisão por cotas) do usuário.



5. O usuário clica em no botão “não”.



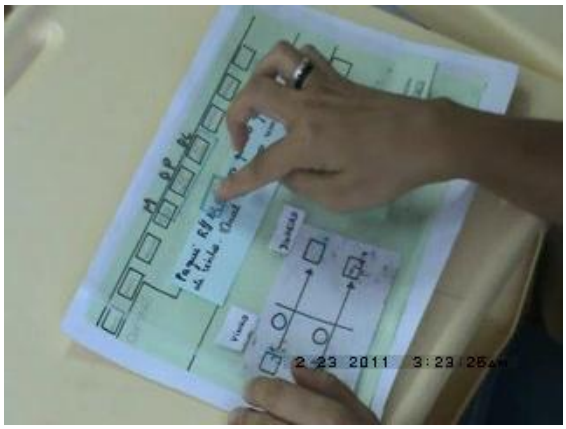
6. A interface exibe novamente a seguinte mensagem para o usuário: *“Escolha no menu a legenda correspondente a categoria a qual pertence à situação-problema”*.



7. O usuário “clica” no botão referente à legenda de Multiplicação, conseguindo assim descobrir a qual categoria pertence à situação-problema.



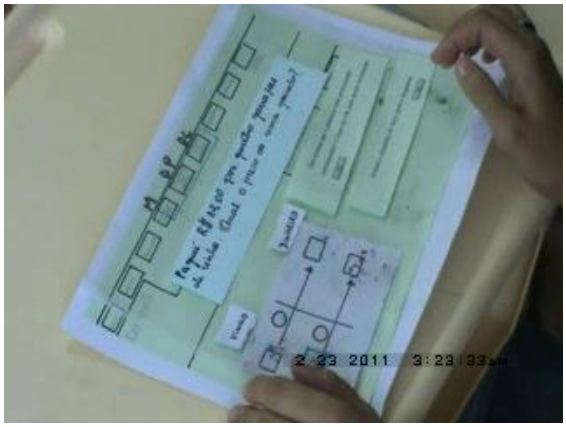
8. A interface exibe o diagrama corresponde à Multiplicação e a seguinte mensagem: “*Arraste os elementos do texto para o diagrama”*. Os elementos são evidenciados através de quadrinhos de seleção.



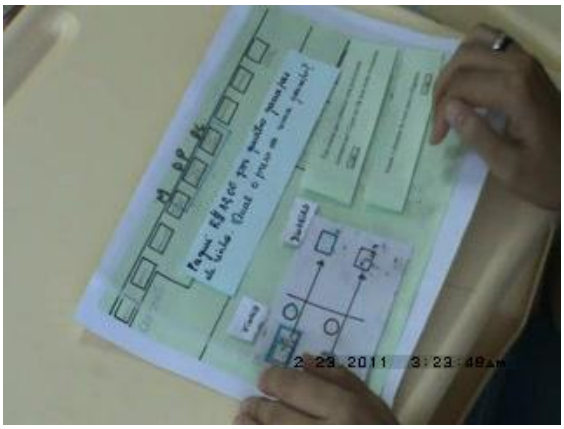
9. O usuário arrasta o número 12 para o lugar que não corresponde à posição correta do diagrama, implicando assim no primeiro erro de posicionamento.



11. O usuário mais uma vez número 12 para o lugar que não corresponde à posição correta do diagrama, implicando assim no segundo erro de posicionamento.



10. Diante do erro cometido pelo usuário, a interface exibe a seguinte mensagem: *"Tem certeza que o elemento foi posicionando corretamente? Clique "ok" para tentar novamente.*



12. A interface novamente exibe a seguinte mensagem: *"Tem certeza que o elemento foi posicionando corretamente? Clique "ok" para tentar novamente.*





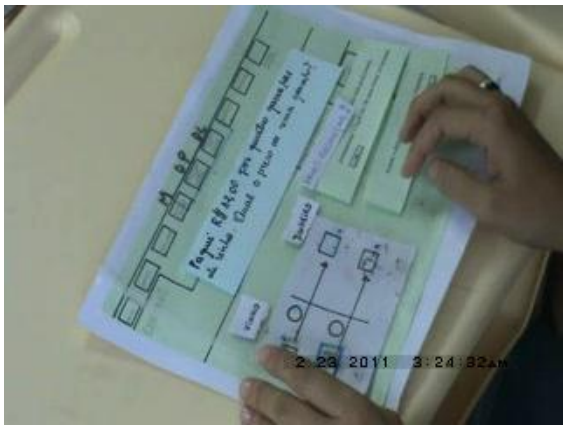


13. Nas três próximas ações, o usuário comete erros consecutivos de posicionamento





15. O usuário arrasta o número 12 para posição corresponde no diagrama.

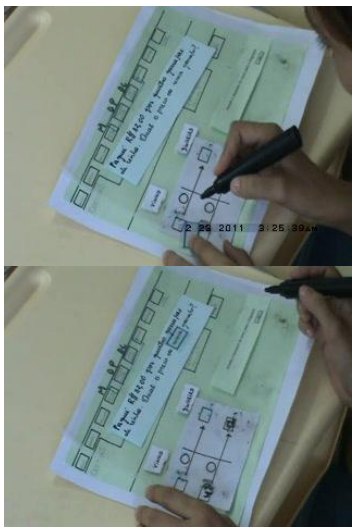


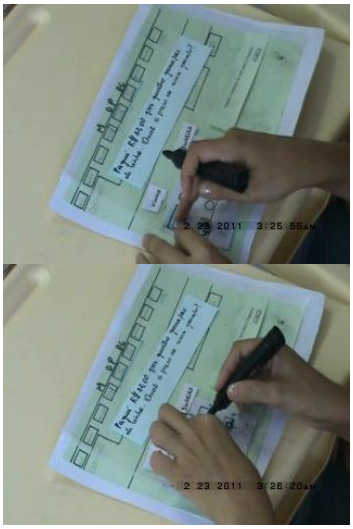
14. A interface exibe então a mensagem: "*Vamos recomeçar?"*



16. A interface identifica o próximo elemento e exibe a seguinte mensagem: "*Arraste os elementos do texto para o diagrama"*. Os elementos são

evidenciados através de quadrinhos de seleção.





17. O mesmo processo é repetido e o usuário consegue compor o diagrama corretamente, fazendo o cálculo no final. Uma mensagem parabenizando o usuário é então exibida.

Anexo B

Cenário dos Casos de Uso da interface GERARD

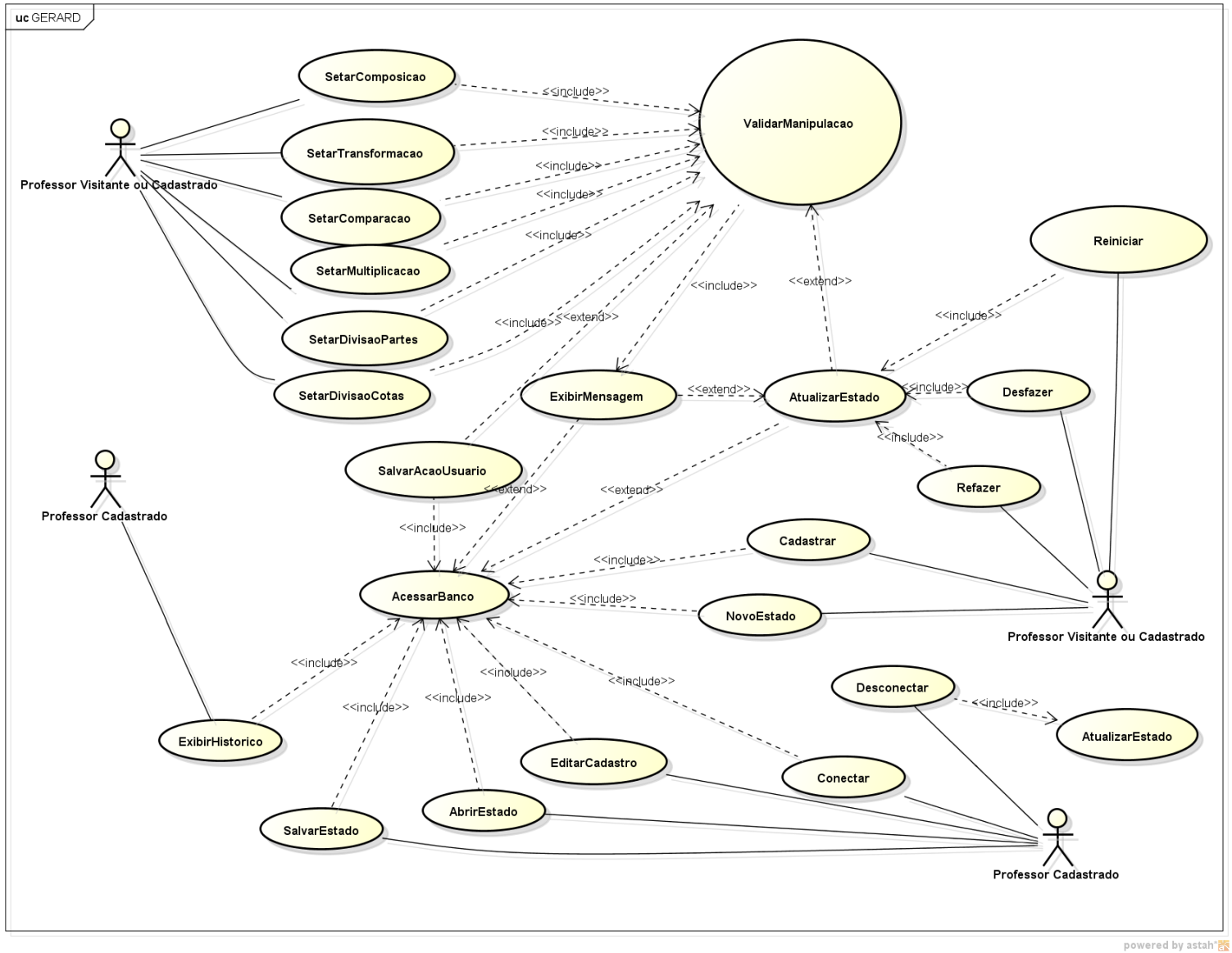


Figura 39. Diagrama de casos de uso da interface educativa GERARD

Atores:

Professor Visitante

Este usuário pode resolver as questões de matemática propostas pelo software a partir da manipulação dos diagramas de Vergnaud.

Ator: Professor Cadastrado

Este usuário pode resolver as questões de matemática propostas pelo software a partir da manipulação dos diagramas de Vergnaud e também salvar seu estado, abrir um estado salvo anteriormente, editar seu dados cadastrados e visualizar seu histórico de utilização do software.

Casos de uso

Caso de Uso:	AbrirEstado			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface para o ultimo estado salvo no banco pelo usuário, caso exista.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.ProfessorCadastrado	*UseCaseLink*	AbrirEstado
	AbrirEstado	*Include*	AcessarBanco

Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona o botão ou item de menu "abrir". 2. O sistema acessa o banco e a interface é atualizada.
Exception **Erro ao abrir**	No passo 2, pode ser verificado que não existe nenhum estado salvo. Uma mensagem é exibida para o usuário.

Caso de Uso:	AtualizarEstado			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011

Descrição:	Atualiza as variáveis para uma nova configuração de estado.		
Associação:	**Fonte**	**Conector**	**Alvo**
	AtualizarEstado	*Extend*	ValidarManipulacao
	AtualizarEstado	*Extend*	AcessarBanco
	Sair	*Include*	AtualizarEstado
	Refazer	*Include*	AtualizarEstado
	Desfazer	*Include*	AtualizarEstado
	Reiniciar	*Include*	AtualizarEstado
	ExibirMensagem	*Extend*	AtualizarEstado

Caso de Uso:	DesfazerEstado			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface para a configuração do estado imediatamente anterior.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.Professor	*UseCaseLink*	DesfazerEstado
	DesfazerEstado	*Include*	AtualizarEstado

Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona o botão ou o item de menu "desfazer". 2. O sistema atualiza as variáveis internas com os valores imediatamente anteriores. 3. O sistema atualiza a interface do aplicativo.

Caso de Uso:	ExibirMensagem			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso exibe mensagens para o usuário mediante intereações deste com o				

	software.			
Associação:	**Fonte**		**Conector**	**Alvo**
	ValidarManipulacao		*Include*	ExibirMensagem
	ExibirMensagem		*Extend*	AcessarBanco
	ExibirMensagem		*Extend*	AtualizarEstado

Caso de Uso:	Reiniciar			Caso de Uso Id:
Status:	Proposed	Versão:	1.0	Fase: 1.0
Autor:	Kecia	Criado:	20/11/2010	Modificado:01/05/2011
Descrição:	Inicializa o software para o estado inicial, fazendo com que o usuário recomece a tarefa.			
Associação:	**Fonte**		**Conector**	**Alvo**
	Ator.Professor		*UseCaseLink*	Reiniciar
	.Reiniciar		*Include*	AtualizarEstado

Caso de Uso:	RefazerEstado			Caso de Uso Id:
Status:	Proposed	Versão:	1.0	Fase: 1.0
Autor:	Kecia	Criado:	20/11/2010	Modificado:01/05/2011
Descrição:	Este caso de uso atualiza a interface para a configuração do estado imediatamente posterior ao atual.			
Associação:	**Fonte**		**Conector**	**Alvo**
	Ator.Professor		*UseCaseLink*	RefazerEstado
	RefazerEstado		*Include*	AtualizarEstado
Cenários:				
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona o botão ou item de menu "refazer". 2. O sistema atualiza as variáveis internas com os valores imediatamente posteriores. 3. O sistema atualiza a interface do aplicativo.			

Caso de	Desconectar	Caso de Uso Id:

Uso:					
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso desconecta o usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.ProfessorCadastrado	*Association*	Desconectar
	Desconectar	*Include*	AtualizarEstado

Caso de Uso:	SalvarAcaoUsuario			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso salva o estado corrente da interface a ação realizada pelo usuário no banco de dados.				

Associação:	**Fonte**	**Conector**	**Alvo**
	SalvarAcaoUsuario	*Include*	AcessarBanco
	. SalvarAcaoUsuario	*Extend*	ValidarManipulacao

Caso de Uso:	SetarComparacao			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface exibindo o diagrama de Comparação de Medidas (representado por uma seta para cima), mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	SetarComparacao	*Include*	ValidarManipulacao
	Ator.Professor	*Association*	SetarComparacao

Cenários:	
Basic Path	1. Este caso de uso começa quando o usuário

Fluxo básico	seleciona a opção "composição de medidas". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface. 5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.
Alternate **Escolha errada do diagrama**	No passo 3, depois de receber o resultado da validação, o usuário pode ter escolhido o diagrama errado, isto é, que não corresponde à situação-problema apresentada. Neste caso: 1. Uma caixa com uma mensagem é exibida: "Na situação-problema existe um conjunto que está sendo comparado em relação a outro conjunto?" 2. O usurário clica no botão "ok" 3. Uma caixa com uma mensagem é exibida: "Escolha no menu a legenda correspondente a qual pertence à situação-problema" 4. O usuário clica no botão "ok". 5. Inicia-se passo 1 do fluxo básico.

Caso de Uso:	SetarComposicao			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface exibindo o diagrama de Composição de Medidas (representado pelo símbolo "abre chave") mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.Professor	*Association*	SetarComposicao
	SetarComposicao	*Include*	ValidarManipulacao

Cenários:	
Basic Path **Fluxo Básico**	1. Este caso de uso começa quando o usuário seleciona a opção "composição de medidas". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface. 5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.
Alternate **Escolha errada de diagrama**	No passo 3, depois de receber o resultado da validação, o usuário pode ter escolhido o diagrama errado, isto é, que não corresponde à situação-problema apresentada. Neste caso: 1. Uma janela com a seguinte mensagem é exibida: "Na situação-problema existem partes que formam um todo?" 2. O usurário clica no botão "ok" 3. Uma janela com uma mensagem é exibida: "Escolha no menu a legenda correspondente a qual pertence à situação-problema." 4. O usuário clica no botão "ok". 5. Inicia-se passo 1 do fluxo básico.

Caso de Uso:	SetarDivisaoCotas			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface exibindo o diagrama de Divisão Por Cotas mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.Professor	*Association*	SetarDivisaoCotas
	.SetarDivisaoCotas	*Include*	Usuario Professor.ValidarMa

	nipulacao
Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona a opção "divisão por cotas". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface. 5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.

Caso de Uso:	SetarDivisaoPartes			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface exibindo o diagrama de Divisão Por Partes mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	SetarDivisaoPartes	*Include*	ValidarManipulacao
	Ator.Professor	*Association*	SetarDivisaoPartes

Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona a opção "divisão por partes". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface.

	5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.

Caso de Uso:	SetarMultiplicacao			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza a interface exibindo o diagrama de Multipicação de Medidas, mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	SetarMultiplicacao	*Include*	.ValidarManipulacao
	Ator.Professor	*Association*	SetarMultiplicacao

Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona a opção "multiplicação". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface. 5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.

Caso de Uso:	SetarTransformacao			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso atualiza o estado do software exibindo o diagrama de Transformação de Medidas (representado por uma seta para a direita), mediante validação da manipulação realizada pelo usuário.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.Professor	*Association*	SetarTransformacao

	SetarTransformacao	*Include*	ValidarManipulacao

Cenários:	
Basic Path **Fluxo básico**	1. Este caso de uso começa quando o usuário seleciona a opção "transformação de medidas". 2. O usuário arrasta o objeto até a área onde ele deseja inserir o diagrama. 3. O sistema valida a manipulação do usuário acionando o caso de uso: ValidarManipulacao 4. O sistema atualiza a interface. 5. Uma mensagem é exibida parabenizando o usuário pela escolha correta.
Alternate **Escolha errada do diagrama**	No passo 3, depois de receber o resultado da validação, o usuário pode ter escolhido o diagrama errado, isto é, que não corresponde à situação-problema apresentada. Neste caso: 1. Uma caixa com uma mensagem é exibida: "Na situação-problema existem um conjunto inicial que se transforma num conjunto final?" 2. O usurário clica no botão "ok" 3. Uma caixa com uma mensagem é exibida: "Escolha no menu a legenda correspondente a qual pertence à situação-problema" 4. O usuário clica no botão "ok". 5. Inicia-se passo 1 do fluxo básico.

Caso de Uso:	ValidarManipulacao			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso valida todas as manipulações realizadas pelo usuário na construção do diagrama de Vergnaud para resolução da situação-problema.				

Associação:	**Fonte**	**Conector**	**Alvo**
	DesfazerEstado	*Include*	Usuario Professor.ValidarManipulacao

SetarComposicao	*Include*	Usuario Professor.ValidarManipulacao
SetarTransformacao	*Include*	Usuario Professor.ValidarManipulacao
SetarComparacao	*Include*	Usuario Professor.ValidarManipulacao
SetarMultiplicacao	*Include*	Usuario Professor.ValidarManipulacao
SetarDivisaoPartes	*Include*	Usuario Professor.ValidarManipulacao
SetarDivisaoCotas	*Include*	Usuario Professor.ValidarManipulacao
ExibirMensagem	*Extend*	Usuario Professor.ValidarManipulacao
AtualizarEstado	*Extend*	Usuario Professor.ValidarManipulacao

Caso de Uso:	NovoEstado			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kecia	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Apresenta para o usuário uma nova situação-problema.				
Associação:	**Fonte**		**Conector**	**Alvo**	
	Ator.Professor		*Association*	NovoEstado	
	NovoEstado		*Include*	AcessarBanco	

Caso de Uso:	SalvarEstado			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011

Descrição:	Este caso de uso salva o estado corrente da interface (diagramas e situação-problema vigentes, etc) no banco de dados.		
Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.ProfessorCadastrado	*UseCaseLink*	SalvarEstado
	SalvarEstado	*Include*	AcessarBanco

Caso de Uso:	EditarCadastro			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso permite que o usuário edite seus dados pessoais.				
Associação:	**Fonte**		**Conector**	**Alvo**	
	Ator.ProfessorCadastrado		*UseCaseLink*	EditarCadastro	
	EditarCadastro		*Include*	AcessarBanco	

Caso de Uso:	ExibirHistorico			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso permite que o usuário visualize seu histórico de utilização, contendo as ajudas, quantidades de erros e as situações problemas utilizadas.				
Associação:	**Fonte**		**Conector**	**Alvo**	
	Ator.ProfessorCadastrado		*UseCaseLink*	ExibirHistorico	
	ExibirHistorico		*Include*	AcessarBanco	

Caso de Uso:	Conectar			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso permite que o usuário cadastrado que o mesmo se conecte ao mediante senha e login.				
Associação:	**Fonte**		**Conector**	**Alvo**	
	Ator.ProfessorCadastrado		*UseCaseLink*	Conectar	
	Conectar		*Include*	AcessarBanco	

Caso de Uso:	Cadastrar			Caso de Uso Id:	
Status:	Proposed	Versão:	1.0	Fase:	1.0
Autor:	Kécia de Moura	Criado:	20/11/2010	Modificado:	01/05/2011
Descrição:	Este caso de uso permite que o usuário se cadastre no software.				

Associação:	**Fonte**	**Conector**	**Alvo**
	Ator.Professor	*UseCaseLink*	Cadastrar
	Cadastrar	*Include*	AcessarBanco

Anexo C

Diagramas de classe da interface GERARD

1. Organização

O software GERARD está organizado em cinco pacotes: model, view, controller, util e banco como apresentado na Figura 1.

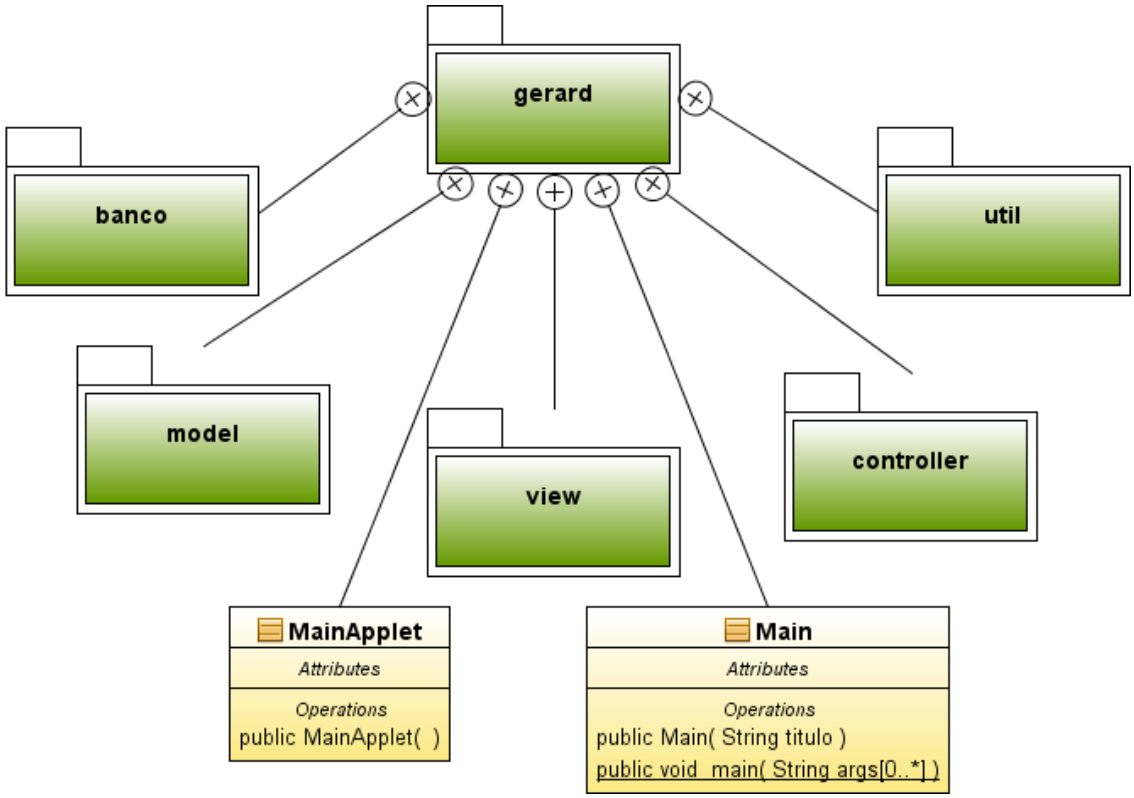


Figura 40. Organização do software

Nas próximas seções, são apresentados os diagramas de classe para cada pacote ilustrado acima bem como para os sub-pacotes existentes.

2. Pacote view

O pacote view contém todas as classes utilizadas para construção das GUIs do software GERARD. O pacote está organizado como mostrado na Figura 2. A visão expandida das classes pode ser observada na Figura 3.

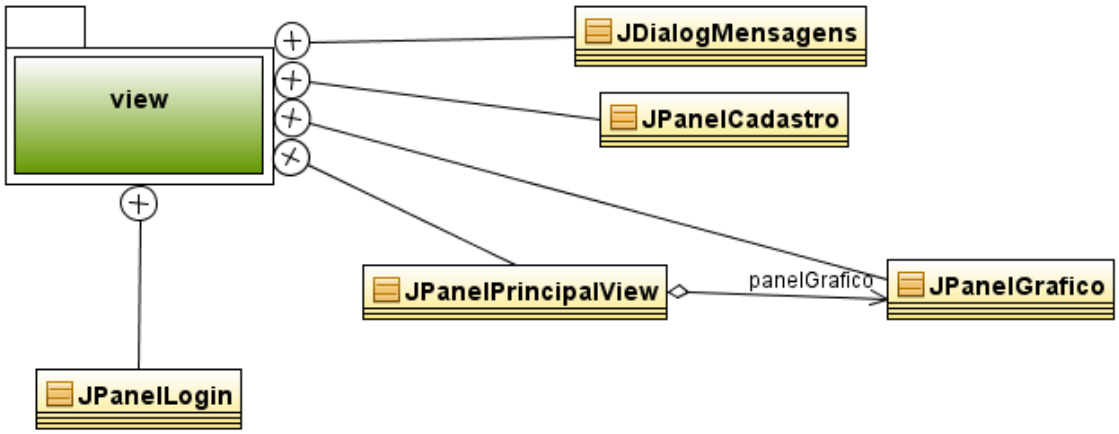


Figura 41. Organização do pacote View.

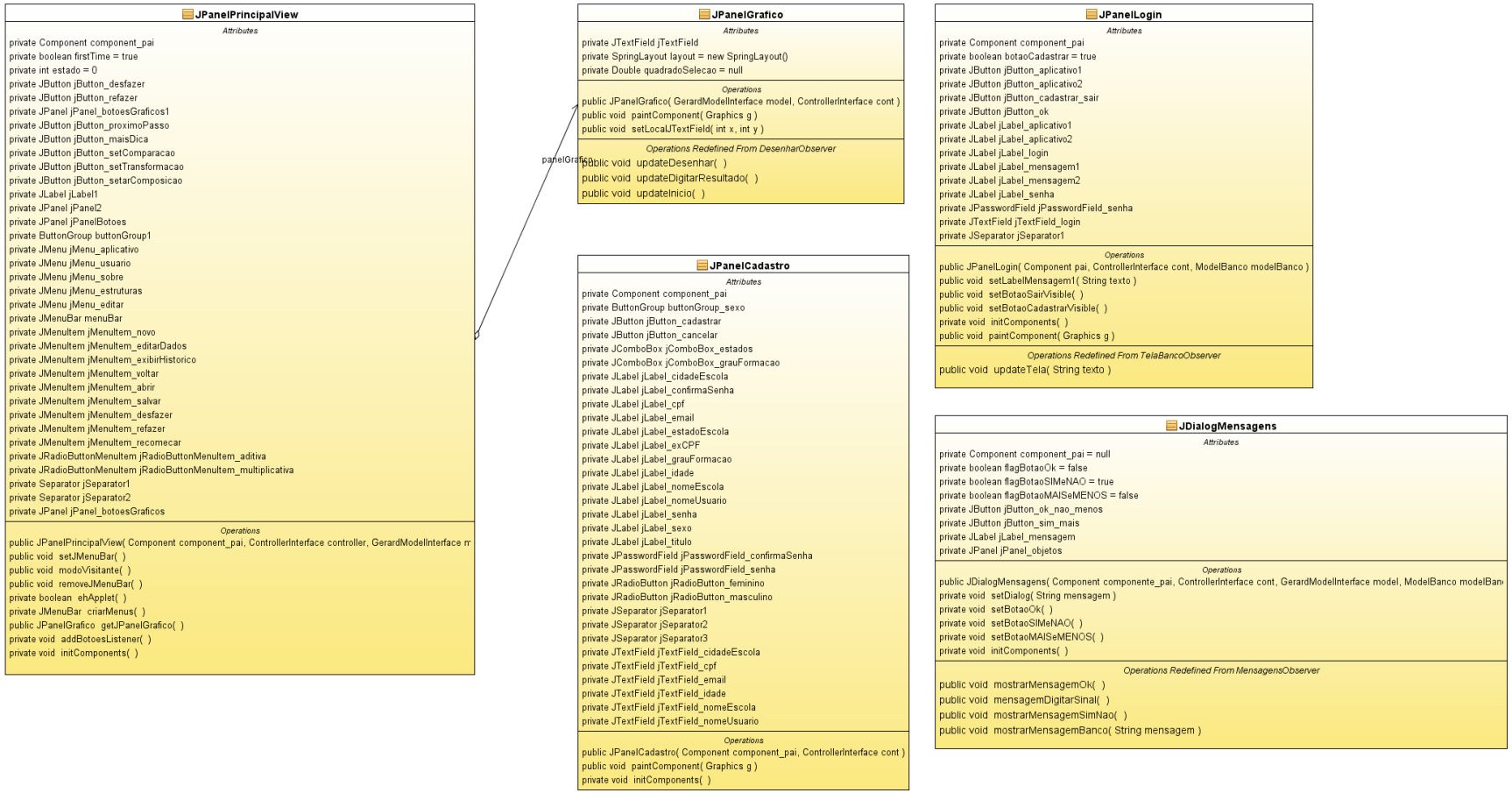


Figura 42. Visão expandida do pacote View.

3. Pacote Model

O pacote Model contém todas as classes utilizadas para implementar a parte lógica do software GERARD. O pacote está organizado como mostrado na Figura 4. A visão expandida das classes pode ser observada na Figura 5.

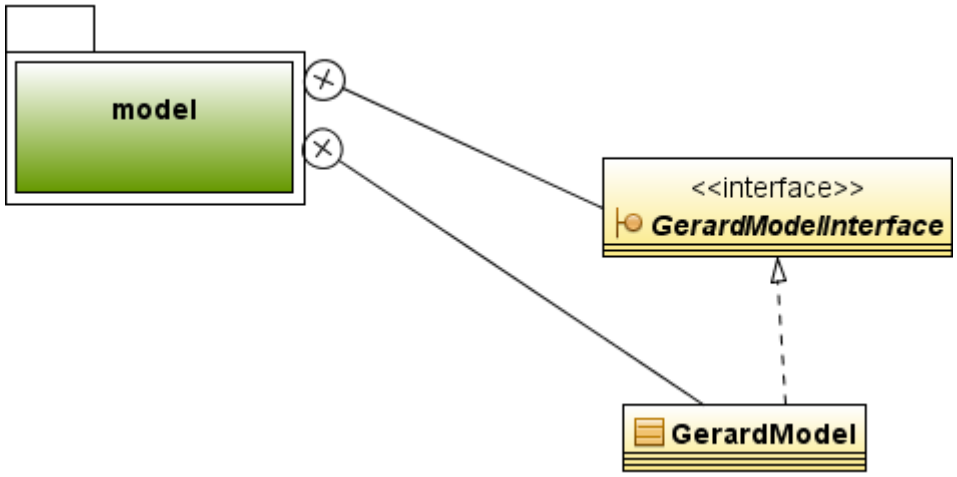


Figura 43. Organização do pacote Model

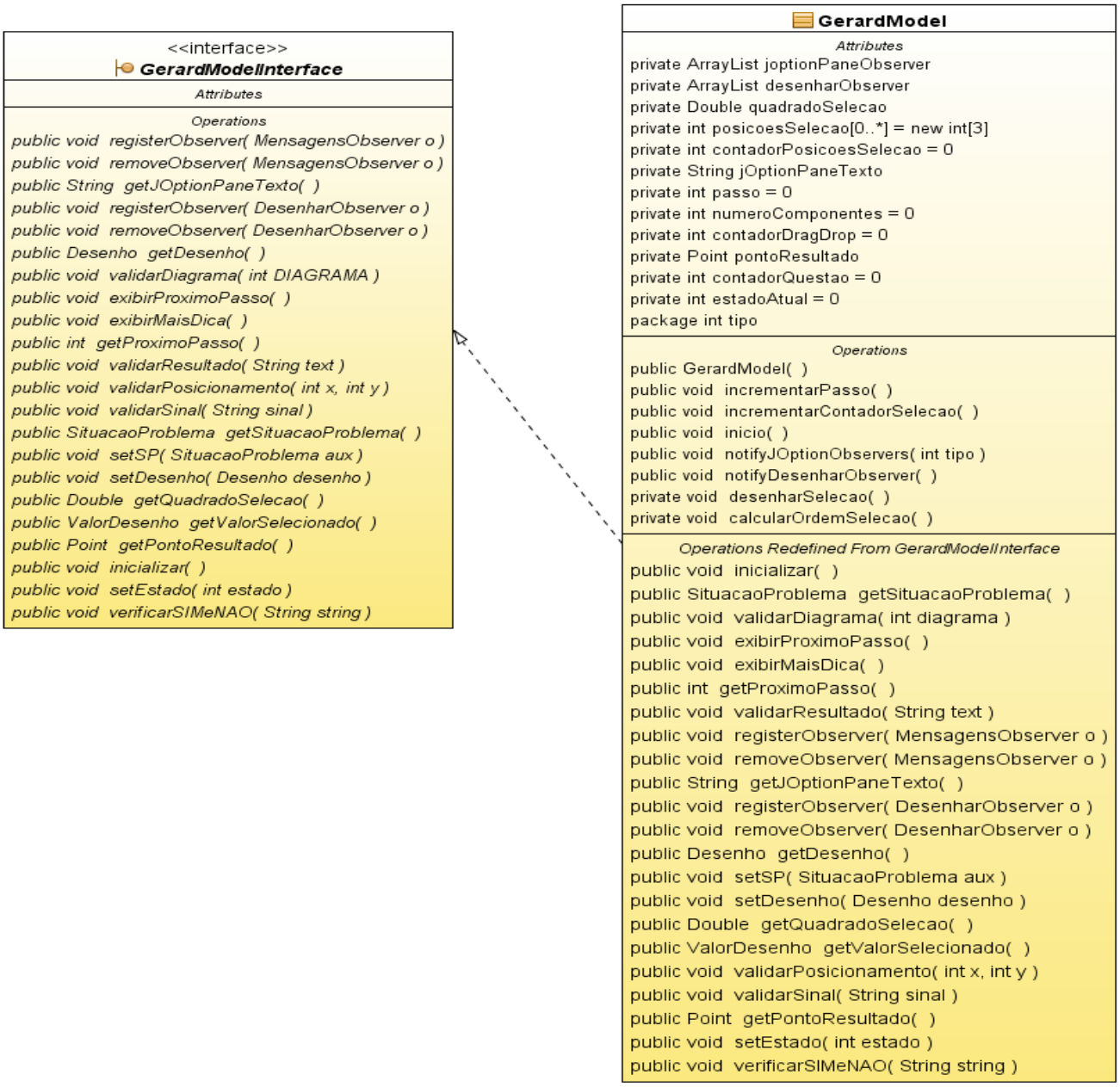


Figura 44. Visão expandida do pacote Model.

4. Pacote Controler

O pacote Controller contém todas as classes utilizadas para implementar a parte de controle do software GERARD. O pacote está organizado como mostrado na Figura 6. A visão expandida das classes pode ser observada na Figura 7.

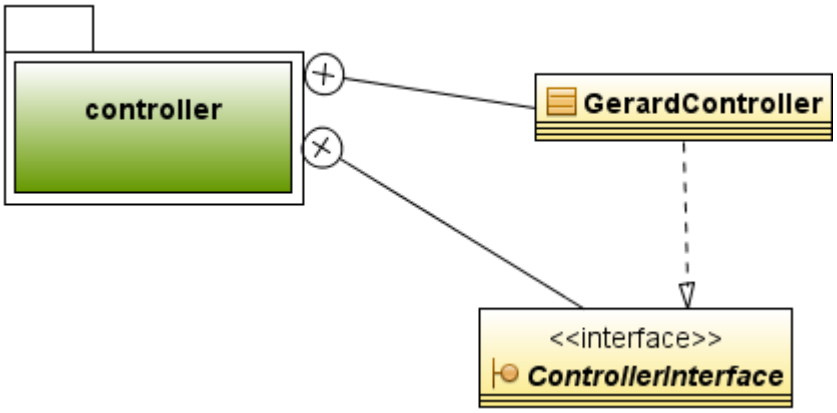


Figura 45. Organização do pacote Controller.



Figura 46. Visão expandida do pacote Controller.

5. Pacote Banco

O pacote Banco contém todas as classes utilizadas para implementar a parte de acesso ao Banco de dados do software GERARD. O pacote está organizado como mostrado na Figura 8. A visão expandida das classes pode ser observada na Figura 9.

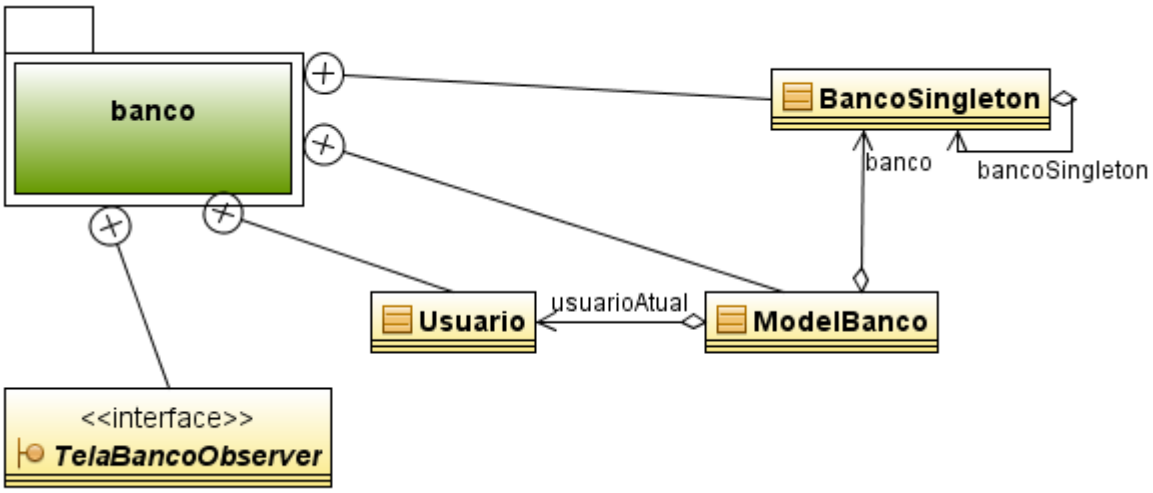


Figura 47. Organização do pacote Banco.

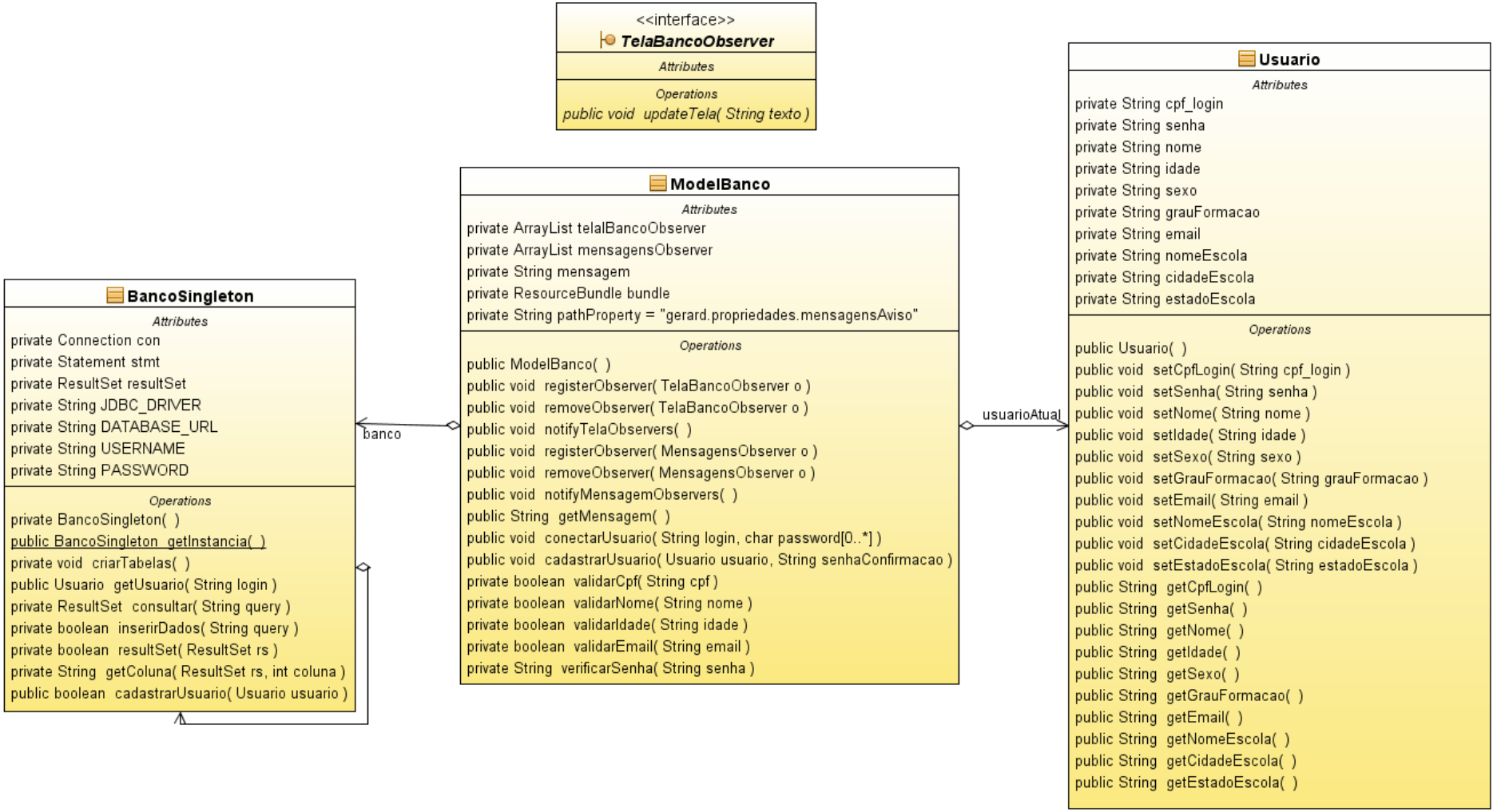


Figura 48. Visão expandida do pacote Banco.

6. Pacote Util

O pacote Util contém todas as classes utilizadas para auxiliar o cálculo lógico, fazer os desenhos gráficos, permitir a implementação em MVC (Model-View-Controller) através das classes “Observers”, entre outros. O pacote está organizado como mostrado na Figura 10. A visão expandida das classes pode ser observada nas Figuras 11 e 12.

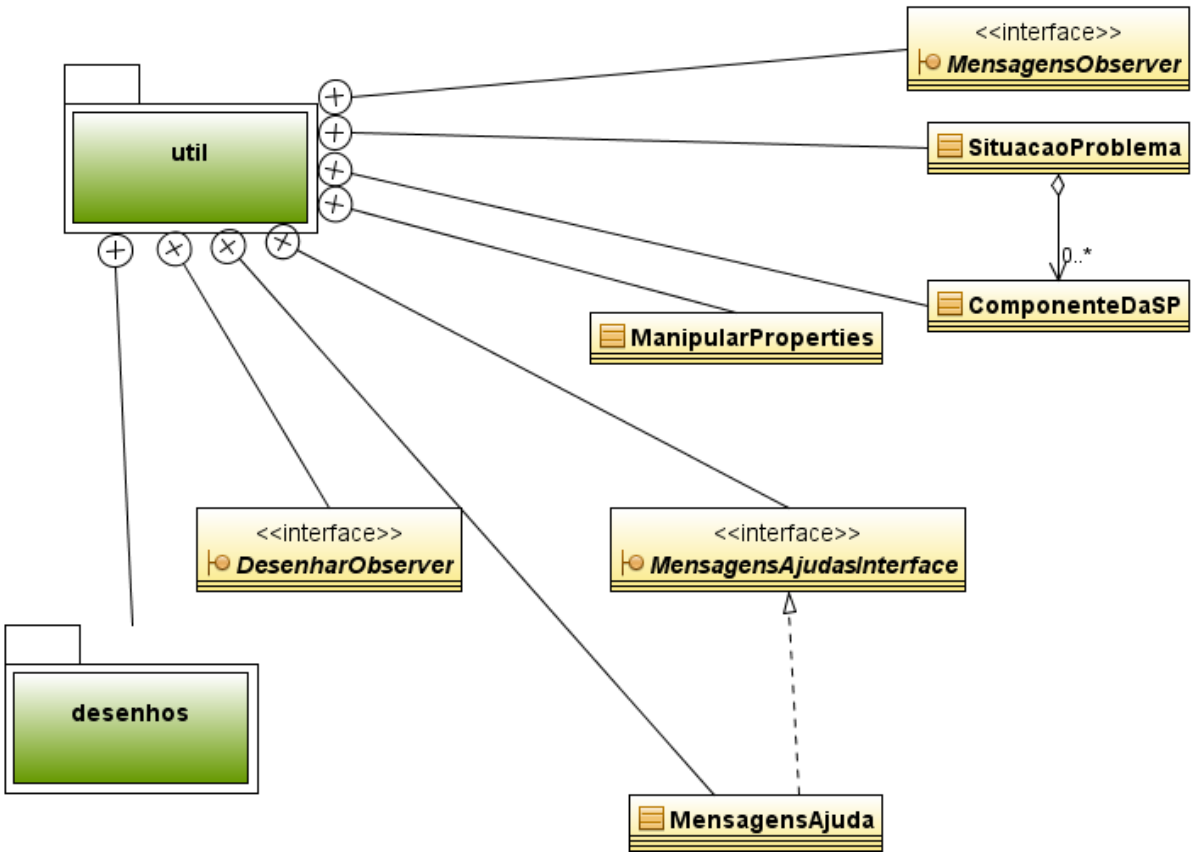


Figura 49. Organização do pacote Util.

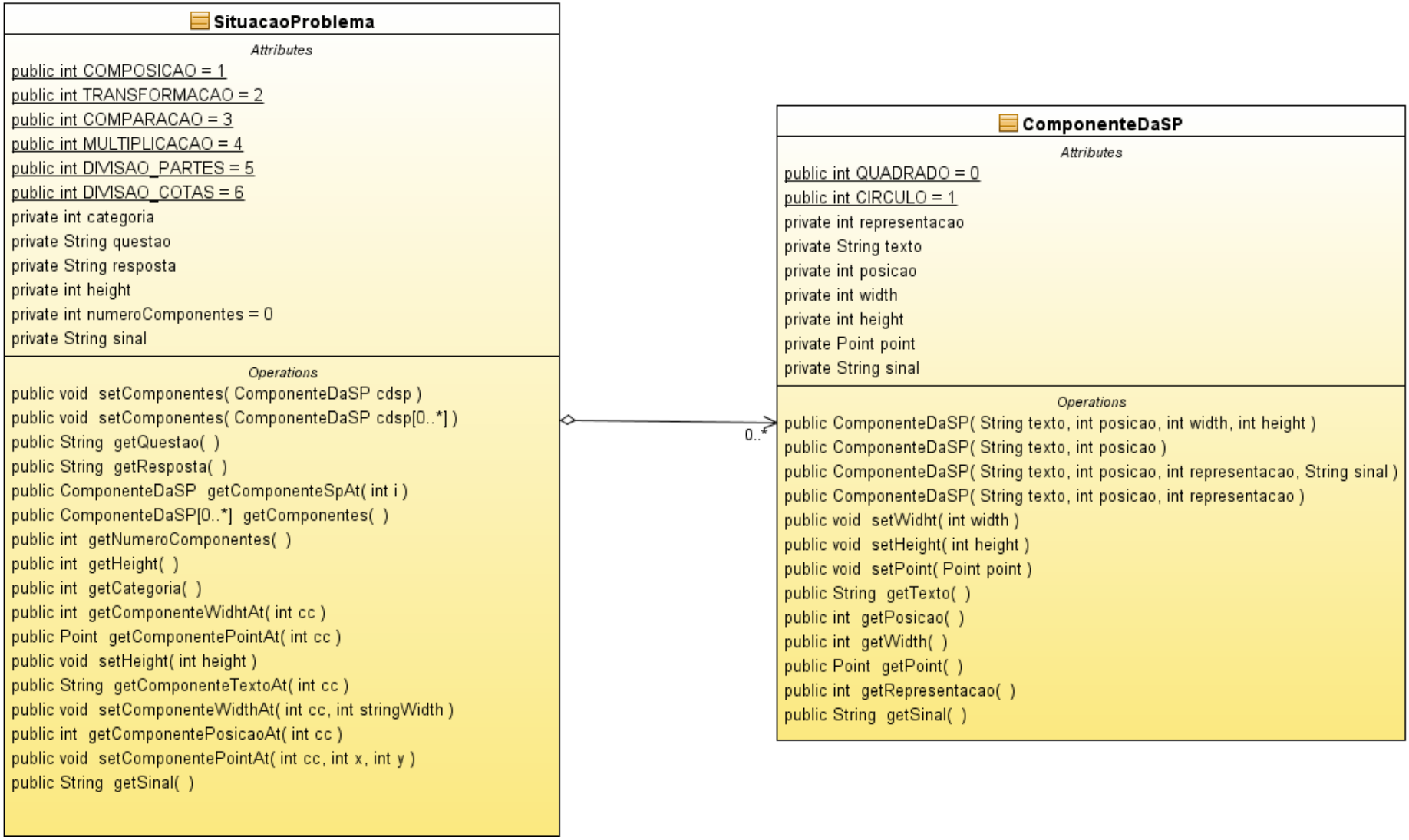


Figura 50. Visão expandida do pacote Util (1).

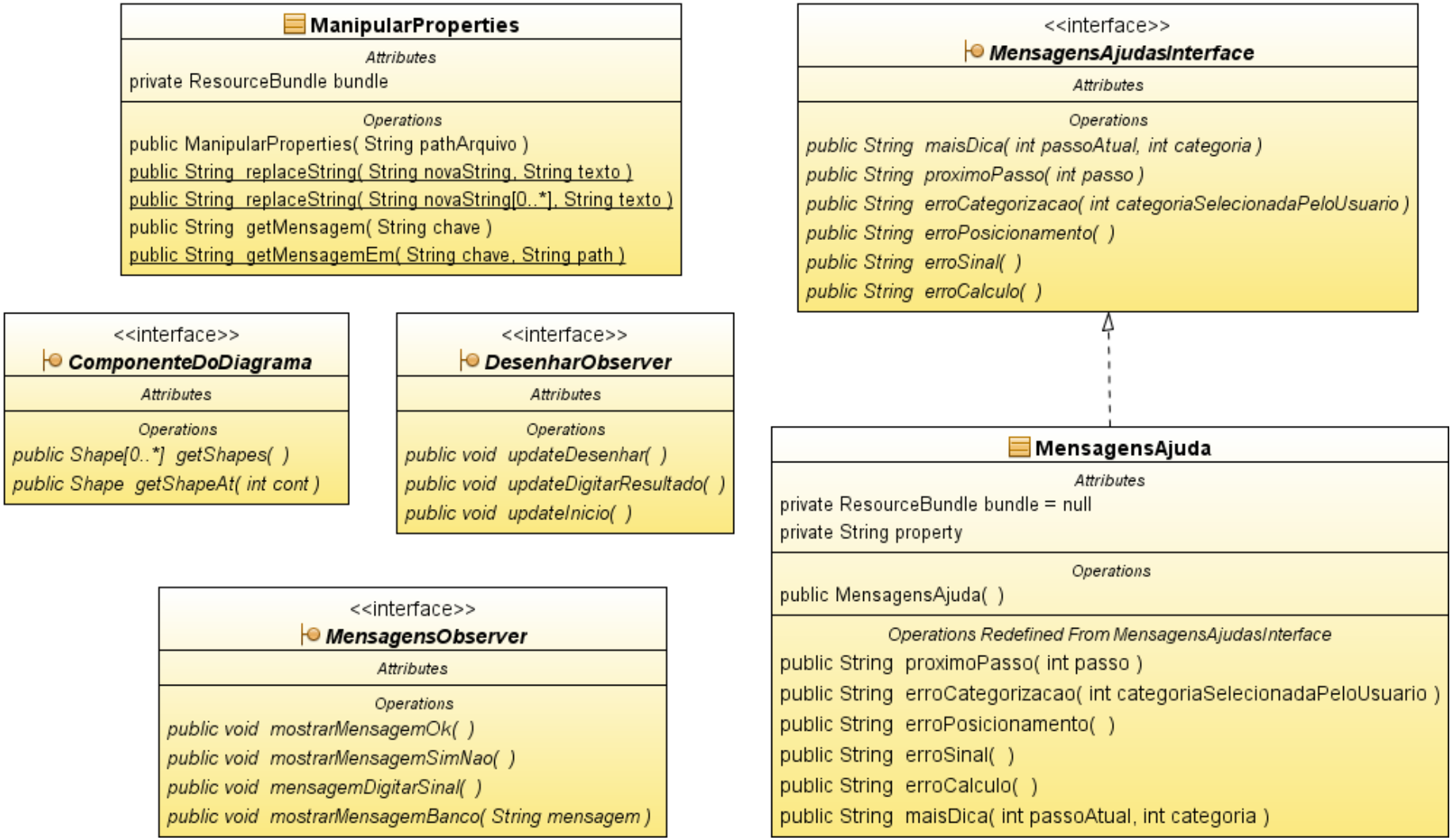


Figura 51. Visão expandida do pacote Util (2).

6.1. Sub-pacote Desenhos

Além das classes, o pacote Util possui um sub-pacote Desenhos que contem todas as classes utilizadas para fazer os desenhos gráficos do software GERARD. A organização do sup-pacote Desenho pode ser visualizada na Figura 13.

Os diagramas do sub-pacote foram modelados utilizando-se o padrão de projeto Decorator. A visão expandida das classes pode ser observada na Figura 14.

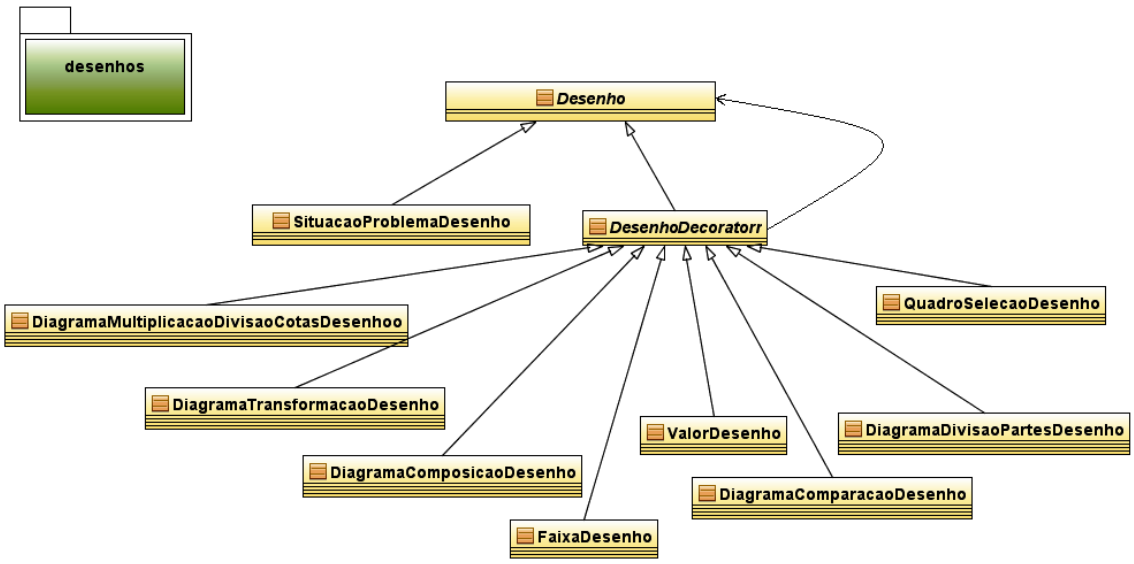


Figura 52. Organização do sub-pacote Util.

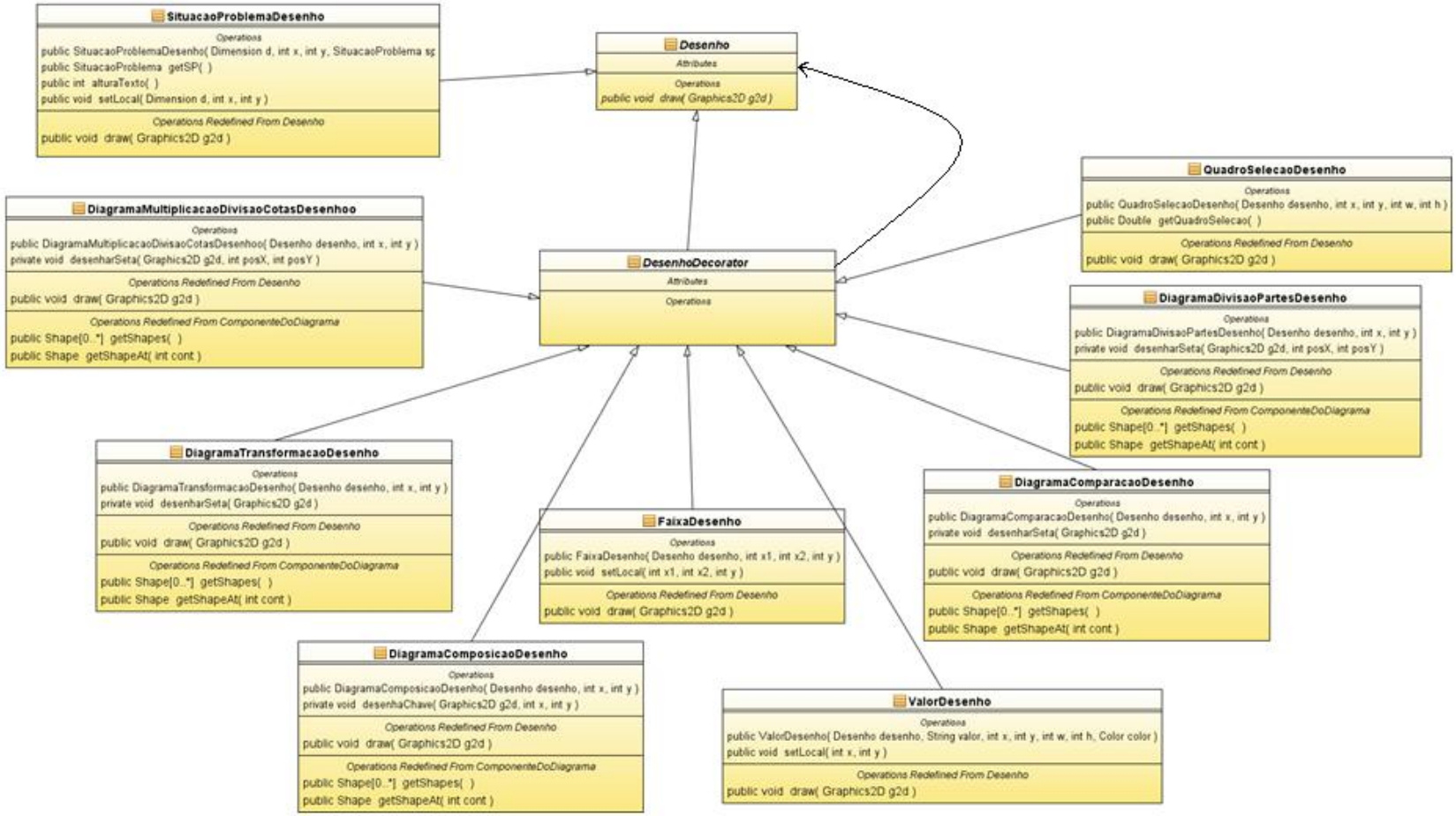


Figura 53. Visão expandida do sub-pacote Desenhos.

7. MVC

O software GERARD foi desenvolvido utilizando-se o modelo MVC (Model-View-Controller). Os relacionamentos entre as classes descritas anteriormente e que evidenciam o modelo são apresentadas nas Figuras 15, 16 e 17.

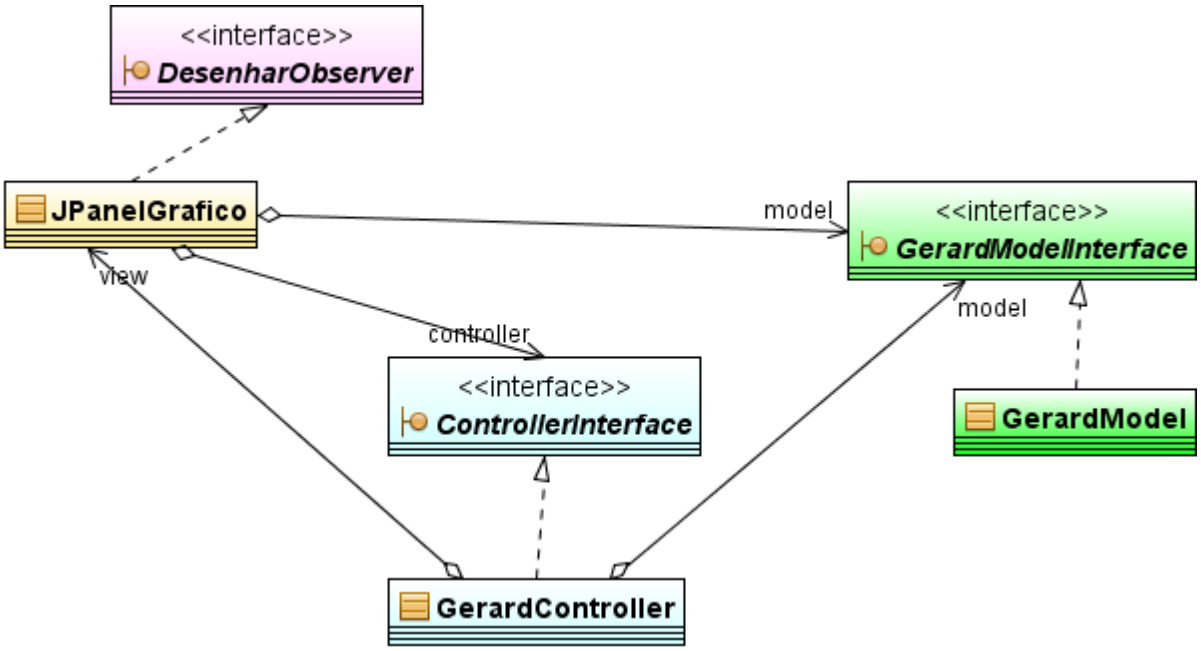


Figura 54. MVC 1.

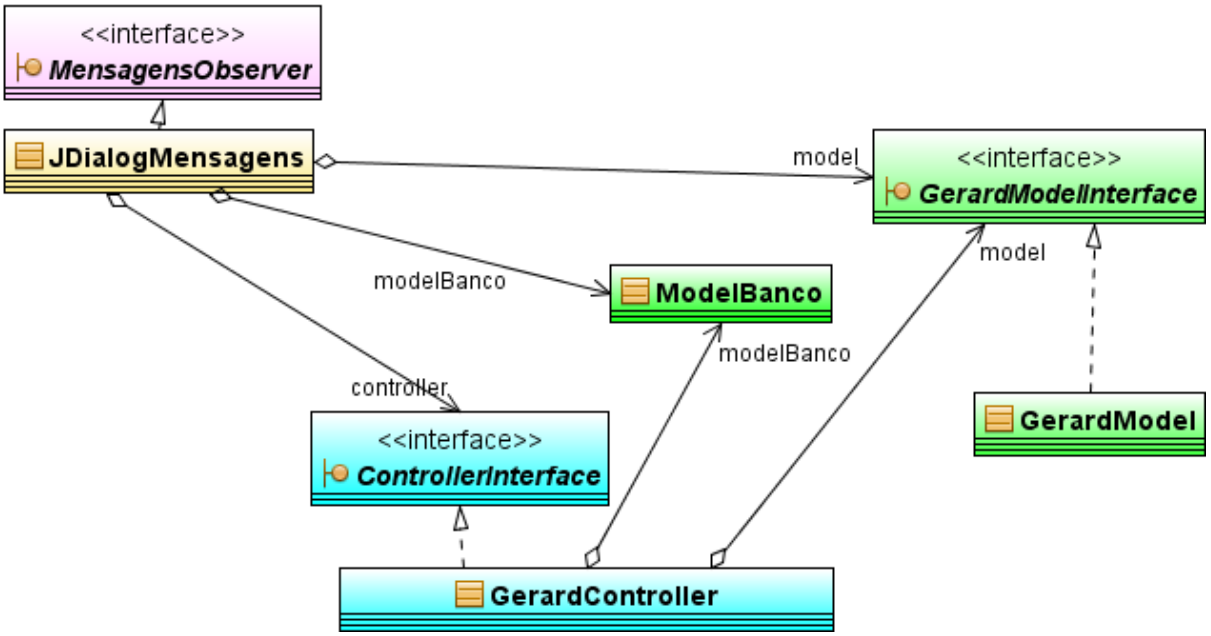


Figura 55. MVC 2.

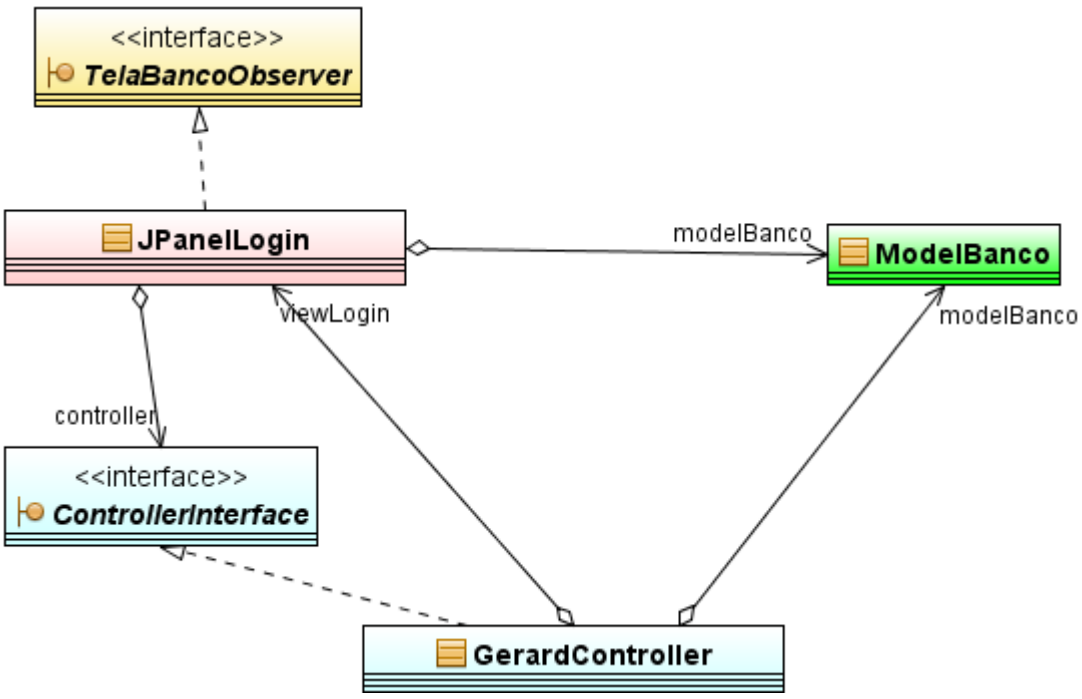


Figura 56. MVC 3.